QAD Enterprise Platform
2020

Developer Guide

QAD Enterprise Platform

70-3391-QEP2020-Rev1

QAD Enterprise Platform

April 2020

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

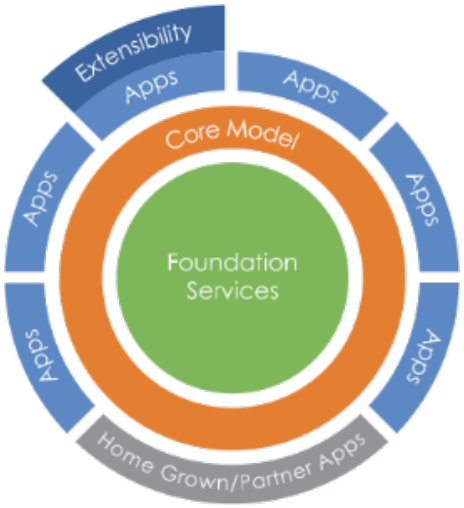
Table of Contents

QAD Enterprise Platform Developers Guide - March 2020

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 element of **QAD Adaptive ERP**, which transforms the user experience, improving business outcomes through more effective users, processes, and decision-making. QAD Adaptive ERP features a new user interface: 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 Web UI, users interact with QAD apps to perform business tasks anywhere, anytime. 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 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 and customers 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)		Typescript tutorials - hackr.io
Chrome Developer Tools	Chrome Dev Tools Tutorial (Anant Agarwal)		
JavaScript	JavaScript Basics		
JavaScript / jQuery	JavaScript & jQuery Tutorials (LittleWebHut)	w3schools on-line Tutorial and Reference	Advanced JavaScript Fundamentals
AngularJS	AngularJS tutorial (Derek Banas)		
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	w3schools on-line Tutorial and Reference	
Chrome Developer Tools	Chrome Dev Tools Tutorial (Anant Agarwal)		Debugging the Web: JavaScript (lynda.com)
TypeScript	Introduction to TypeScript (Jess Chadwick)		Typescript tutorials - hackr.io
AngularJS	Up and Running with AngularJS 1		

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 QAD Enterprise Platform can be used to extend QAD Adaptive ERP in the following ways:

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

Architecture

This section provides an overview of the architecture:

Additionally, see also:

QRA Architecture

QAD Adaptive ERP features 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). Business components for a specific business purpose can be combined into **business documents**, which consist of a *parent* business component with one or more *child* business components.

QAD Adaptive ERP emphasizes user experience through the careful design of user interface metaphors for interacting with the Web UI. QAD carefully limits the number of metaphors in order to assure a cohesive, consistent, and easy-to-learn user experience. In general, the Web UI's "pages" or "screens" are called views. Views can consists of browses, forms, a browse combined with a form (called a hybrid view, or hybrid browse), reports, and so on.

The system itself is installed, configured, and managed using the **Your Application Builder (YAB)** framework, a command-line based tool for system administrators.

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.

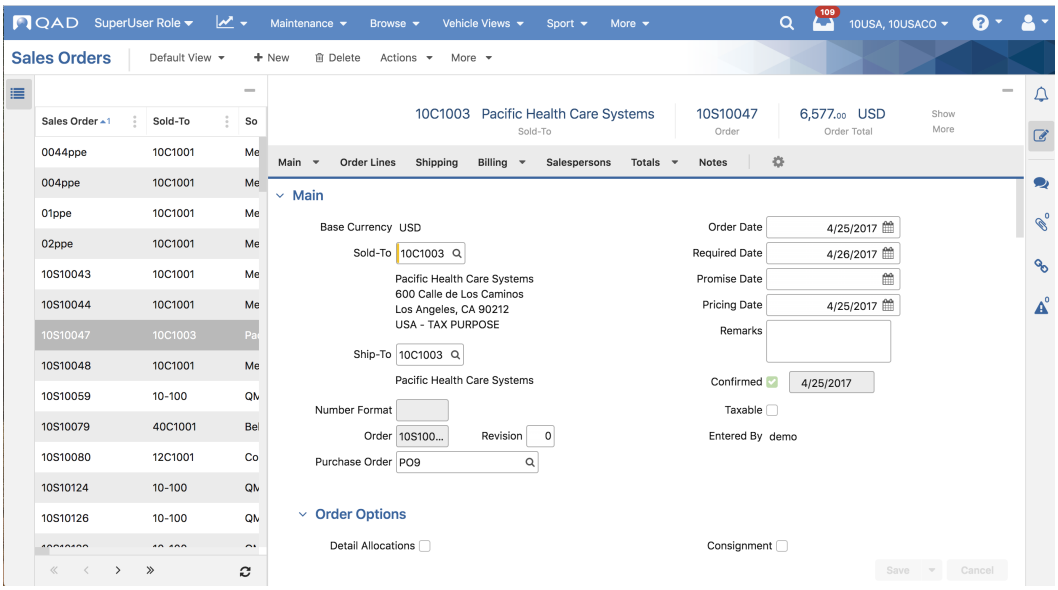
Business components provided by QAD (that is, business components in the "com.qad" environment namespace) cannot be modified.

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 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.

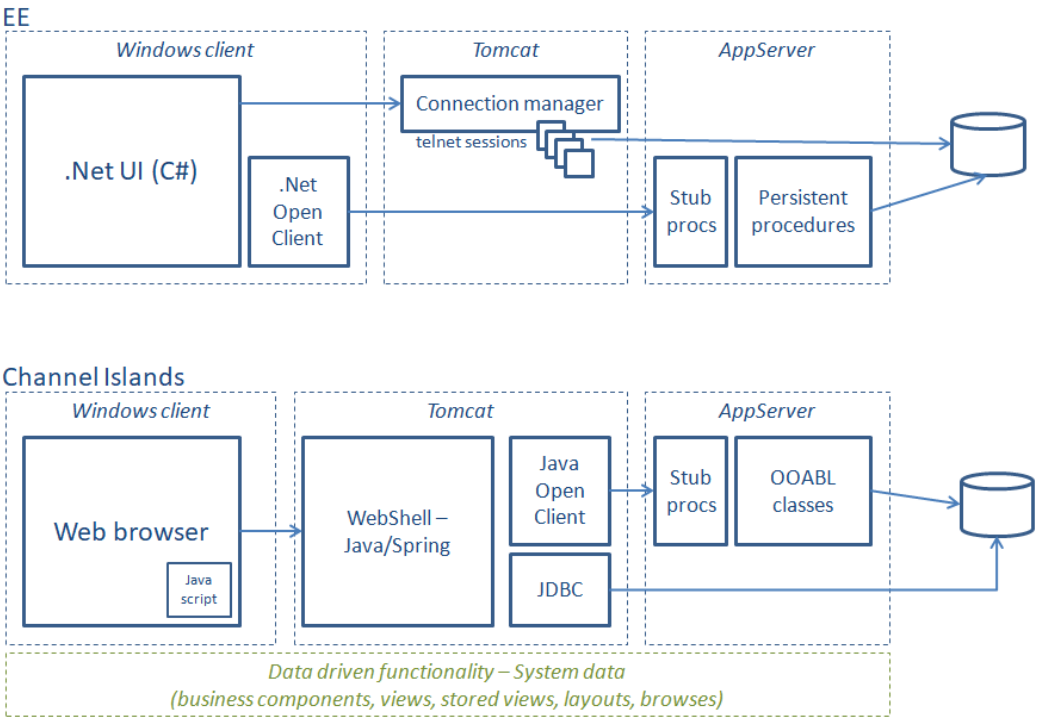
Next, get started with App Development.

Evolution of QAD Architecture

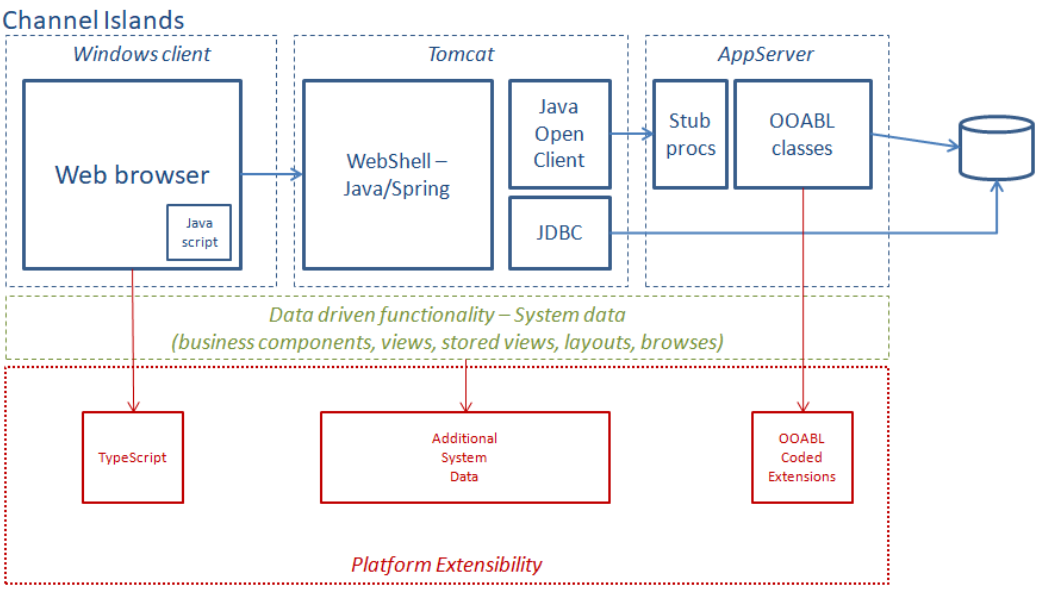
The Channel Islands initiative introduced a new architecture, the QAD Reference Architecture (QRA). The following compares the Enterprise Edition (EE) architecture with the Channel Islands architecture, and shows how the QAD Enterprise Platform can extend Channel Islands.

EE and 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



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

- **BC** — Business Component
- **BD** — Business Document
- **BL** — Business Logic
- **OOABL** — Object Oriented Advanced Business Language (Progress development language)
- **TS** — TypeScript
- **UI** — User Interface
- **URI** — Uniform Resource Identifier
- **URN** — Uniform Resource Name
- **YAB** — Your Application Builder

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 (embed) 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 components 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.
- App Registration. In order to ensure app names are unique across QAD, partners, and customers, apps that intend to be shared or be included in an environment with other shared apps need to be registered with QAD to ensure a unique app registration code. This unique app registration code will also become the prefix for any of the app's business component physical table names. (Note: Coded business components from QAD will display as released and do not need to be registered.) Registering your app is an option available on the Apps view. If you intend to share your app, you should get an app registration code when you create the app.

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 *platform* business components:

> **Coded business components** are business components that are defined with the traditional development process. They do not rely on the Platform's Business Component builder, and they retrieve the metadata from annotations in the code.
>
> **Platform business components** are business components that are defined in the Platform's Business Component builder, and later deployed. These business components do not have any manual code behind them and are completely data-driven. A special kind of platform business component are the *embedded* business components. These are defined in an App for the sole purpose of extending the data for another business component. Embedded business components cannot be run stand-alone.

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

Extension vs. embedded

What's the difference between the terms “extension” and “embedded”?

- Embedded — An attribute of a business component that indicates (1) the business component will be a child BC, and (2) the data will be saved in the same transaction as its parent. The child cannot exist on its own without the parent.
- Extension — This term was introduced by marketing early on in the Platform evolution to replace the term "customizations". It is really anything that's changed on top of QAD functionality (or any namespace extending a different namespace). Extensions are a way for customers to make "changes" to the QAD apps (or any app they do not own) without breaking upgradability. Overriding field properties is a form of BC extension, for example.

Business component fields

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 Relationship

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

Relationships between business components can be used to generically add behavior to the system. The following types of relationships can be distinguished:

- **Lookup relationship**. This relationship 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 "Many-to-one" cardinality (for example, a SalesOrderLine has a field "ItemCode", which drives the Many-to-one BC relationship 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.
- **Child-parent relationship**. This relationship indicates a closer connection between two business components. 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 child-parent related BCs are on one form, we also allow business components 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.
  Note that starting with the September 2019 release, you can create relationships only from the Child side.

The platform supports the extension of data for a given BC. This is done using parent-child relationships. These can have a "One-to-one" or "Many-to-one" cardinality. The BC relationship in this case is always between a business component and an embedded business component. For the "One-to-one" relationships, the system will automatically add these fields to the BC views and BC browses. For the "Many-to-one" relationships, 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.

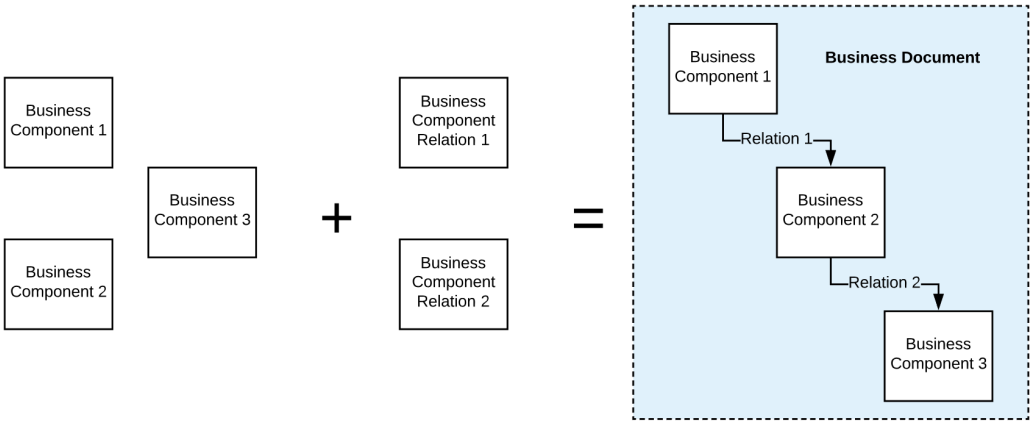
In general, a business component browse can have:

- Fields from the top-level table of the BC
- Fields from the top-level table of a BC that is related 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 relationship to the BC represented in the browse

Business Document

A business document is a collection of related business components, mainly used for inbound integration with the application. The relationships involved are such that they make logical sense to be grouped together as a single business component. This is always expressed via a parent-child relationship. 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 (also called a hybrid view) provides the most common way to interact with a business component (BC) from the Web UI.

The hybrid browse consists of:

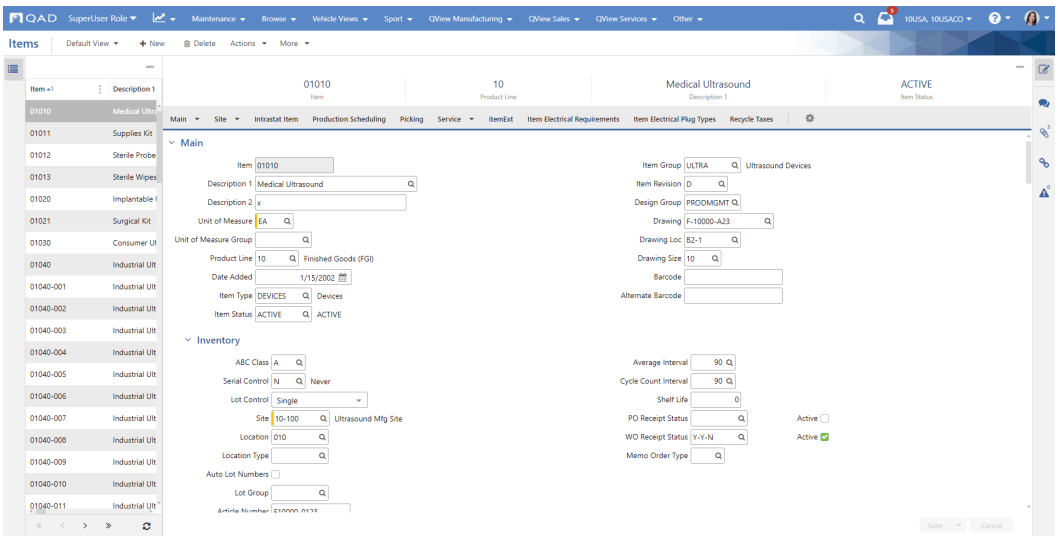
- a **browse** in which every line represents a BC instance (for example, a customer with code "10-100" is an instance of the Customer BC)
- a **form** that appears when a line in the browse is selected by double-clicking, or the user clicks "Edit" or "New". This form represents the BC and visualizes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

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

- The form view can be defined in the form 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 two 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.



Schema Mapping

For each business component, the developer has the capability to link to a table and fields in the physical schema of a data store. We often refer to this as the **Schema Mapping**. Because many of the tables in the traditional mfgpro database can be cryptic, we use the schema mapping in the business components that provide easier to understand names.

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 QAD 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

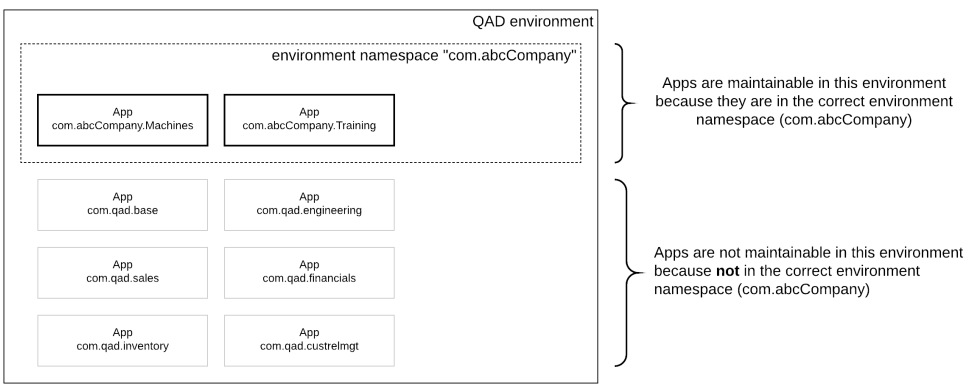
The official way to set the Environment Namespace value is using the yab tooling (`yab update`), providing following properties in the `build/config/`**`configuration.properties`** file. This step needs to be done upon the initial set up of the system.

```
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 relationships, 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 set up 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

This section provides a complete overview of all possibilities of the QAD Enterprise Platform. The pages are intended for use by developers, but are useful for anyone to get an understanding of the capabilities offered by the Platform itself.

App Development Tools and Resources

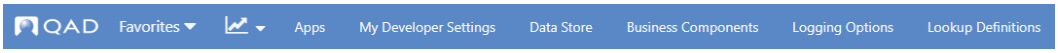
This section introduces all tools that are important for a developer when extending or adding new functionality to the system. The tools support all aspects of the development, packaging, deployment, and debugging of apps. These pages should be used as reference material for each of the tools. In subsequent 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
- My 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:

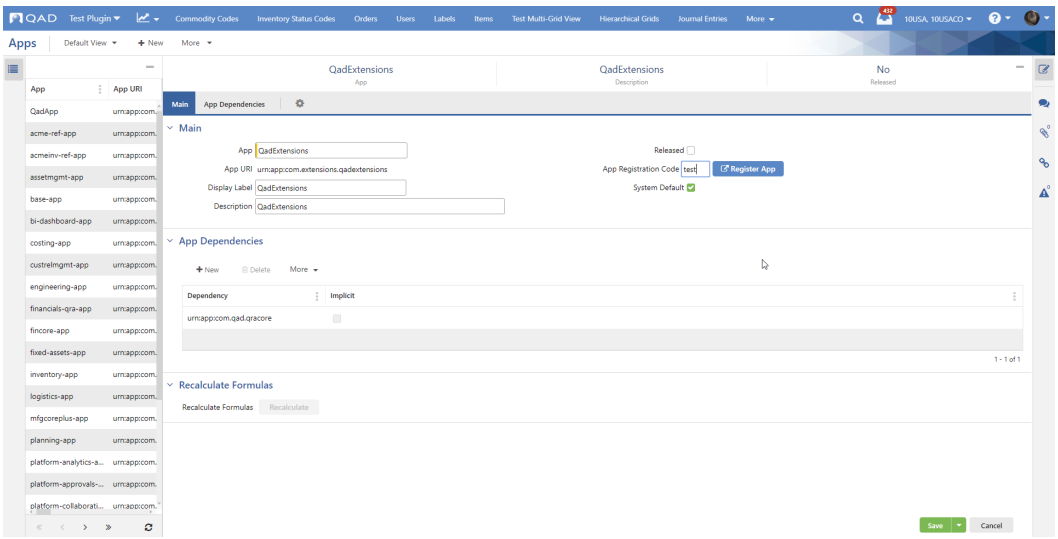
- KPIs view

Apps view

Introduction

With the Apps view, you can create, modify, and delete apps and app dependencies, as well as register your apps with QAD.

Link: App



What is an App?

An App is a 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. Relationships
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 b usiness components for Sales Order, then your app is dependent on the components in the Sales app.

Some dependencies are automatically recognized via business component relationships, we call these "implicit" relationships. If you add a relationship to a business component of another 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. By default, this field displays the value from the App field.

Description

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

Released

Indicates whether an app is released. Releasing an app cannot be undone. Once you select and save this checkbox, both the Released checkbox and the App Registration Code field become disabled.

Note: Coded business components from QAD are displayed as released and do not need to be registered. The Released checkbox and the App Registration Code field are disabled for coded business components.

App Registration Code

Click Register App to register your app with QAD. In order to ensure app names are unique across QAD, partners, and customers, apps that intend to be shared or be included in an environment with other shared apps need to be registered with QAD to ensure a unique app registration code. This unique app registration code will also become the prefix for any of the app's business component physical table names. Coded business components from QAD are displayed as released and do not need to be registered. After registering your app, enter a 4-character code from the QAD App Registration page in the App Registration Code field here. For more information about registering your apps, see App Registration.

System 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 the My 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.

Recalculate Formulas panel

Adding a new Formula field to already deployed BC will undeploy this BC and you will need to deploy it manually again. If there are any existing records for this BC when a new Formula field is added, they should be recalculated by CalculationEngine (to have the value of the newly introduced Formula field stored in the database for those records). This is done automatically when you re-deploy that BC. If you add this newly introduced Formula field to the Browse, the values for this field might not appear there for some time (until the calculations are finished). If, for some reason, you want to force Formula fields recalculation, click the **Recalculate** button. This will recalculate formulas for all business components related to this app.

App Registration

Introduction

In order to ensure app names are unique across QAD, partners, and customers, apps that intend to be shared or be included in an environment with other shared apps need to be registered with QAD to ensure a unique app registration code. This unique app registration code will also become the prefix for any of the app's business component physical table names. Coded business components from QAD are displayed as released and do not need to be registered. Registering your app is an option available on the Apps view. If you intend to share your app, you should get an app registration code through the QAD Store when you create the app.

Deciding to Register an App

Registering your app is highly recommended. Your app must be registered if you want to publish your app in the QAD store. Registering your app also ensures that your app will never conflict with other apps you install in the future. Apps built only for internal testing or training purposes do not necessarily need to be registered.

If after creating an app and its business components, you then decide the app needs to be registered, you will need to run a YAB command. If you are a cloud customer, you cannot directly access YAB, and so you will need to enter a ticket for QAD to run the YAB command. Therefore, for cloud customers, it is best to decide in advance whether to register the app.

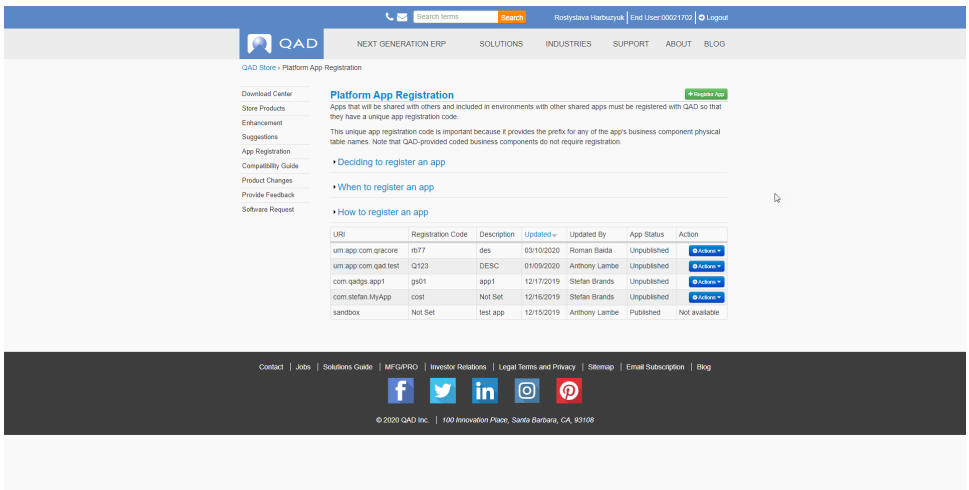
When to Register an App?

QAD strongly recommends that you register your app before creating and deploying any business components in your app. Although it is possible to register your app later, doing so will require deleting any existing data in your database tables, and then running a YAB command. QAD cloud customers will need to enter a ticket for QAD to run the YAB command.

Registering an App Before BC Deployment – Step by Step

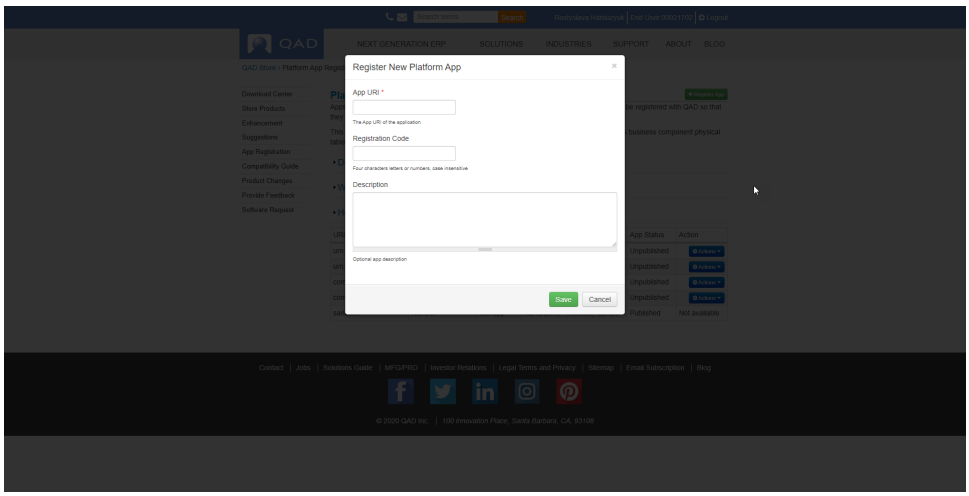
To register an app **before** there are deployed business components in the app, follow these steps:

1. Open the **Apps view** from the menu search, and then open your app.
2. Click the **Register App** button next to the **App Registration Code** field.
3. On the QAD Store > Platform App Registration page, click the green **+ Register App** button, located in the upper-right corner of the page.

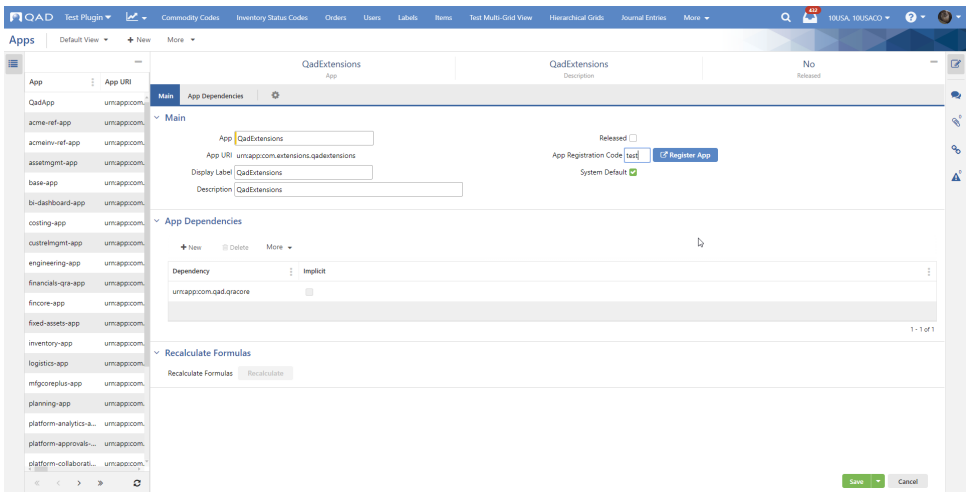


4. Enter the App URI of your application (copy and paste it from the **Apps view**), a unique registration code of your choosing (four characters, case insensitive), an optional description, and then click **Save**. The app registration code must start with a letter and can only contain letters and numbers. Try to use a short but meaningful name for your app registration code. For example, if your company's stock is traded on the market, consider starting

with the ticker symbol representing your company.



5. Go back to the **Apps view** in the application, and in the **App Registration Code** field, enter the registered app code, and then click **Save**.



6. Now, you can proceed with the app development. All BC physical tables are prepended with the app registration code.

Important: Before an app is released, you can change the app registration code. However, when changing the app registration code, all business component physical tables in this app are renamed, and all data in those physical tables are deleted.

Registering an App After BC Deployment – Step by Step

To register an app **after** there are deployed business components in the app, follow these steps:

Prerequisites

- Check is there are any app's business components in the Suspended status. Before saving the app registration code, suspended business components must be reverted to the Initial status by running the Obsolete Schema Start YAB command: yab stop database-extension-obsolete-schema start. Cloud customers need to enter a ticket to run the YAB command.
- The app registration process will delete any existing data in the business component tables. For any data that needs to be saved, export the data from those tables before continuing. Cloud customers need to enter a ticket for QAD to export the data.

Steps

1. Open the **Apps view** from the menu search, and then open your app.
2. Click the **Register App** button next to the **App Registration Code** field.
3. On the QAD Store > Platform App Registration page, click the green **+ Register App** button, located in the upper-right corner of the page.

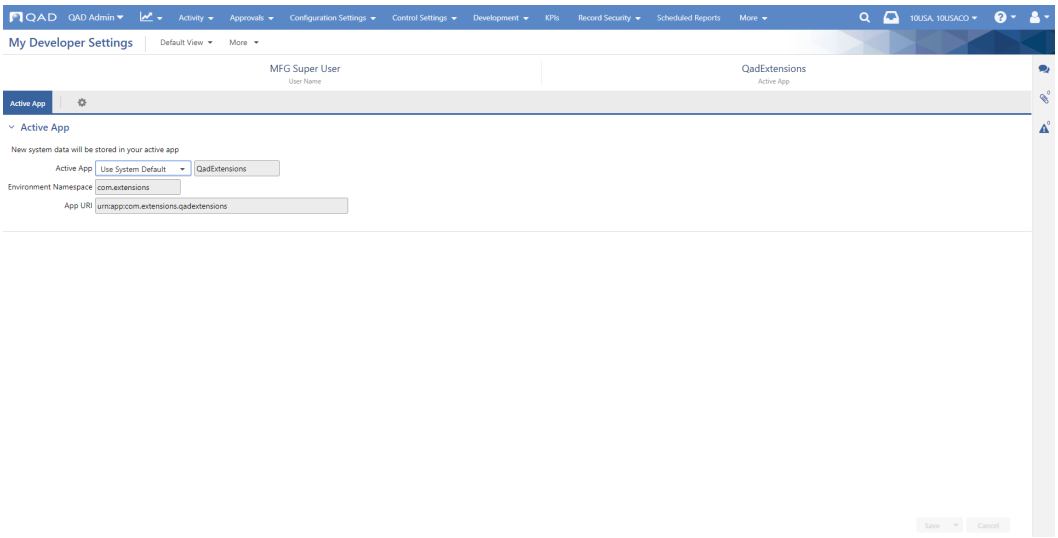
4. Enter the App URI of your application (copy and paste it from the **Apps view**), a unique registration code of your choosing (four characters, case insensitive), an optional description, and then click **Save**. The app registration code must start with a letter and can only contain letters and numbers. Try to use a short but meaningful name for your app registration code. For example, if your company's stock is traded on the market, consider starting with the ticker symbol representing your company.
5. Go back to the **Apps view** in the application, and in the **App Registration Code** field, enter the registered app code, and then click **Save**.
6. In the confirmation message, click **Continue**. Adding an app registration code requires the physical tables to be renamed for all business components in the app. This process will:
    - Suspend all deployed business components in the app until the necessary YAB command is run.
    - Require the App Registration Code YAB command to be run to apply the new app registration code.
    - Delete any existing data in the business component tables. For any data that needs to be saved, export the data from those tables before continuing.
7. Business Components within the app are suspended, but the app registration code is not yet applied. Now, the YAB command is required. Run the App Registration Code YAB command: yab stop database-extension-obsolete-schema start platform-systemcheck. As part of this YAB command, the app's business components are reverted to the Initial status, physical table names are updated, and the business components are re-deployed. Cloud customers need to enter a ticket to run the YAB command.
8. Optionally, load the exported data. Cloud customers need to enter a ticket to load data.
9. Now, you can proceed with the app development. All BC physical tables are prepended with the app registration code.

My Developer Settings view

Introduction

In My 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 relationships that the developer creates using the Business Components view. This also applies to any other platform function that is used to store System Data (s uch as the Form Builder, any stored views, or form layout changes using the Design Layout option).

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

UI Quick Reference

Active App panel

Active App

The name of the currently active app for your development. New system data will be stored in your active app. You can select a system default app or a custom app:

- Use System Default. Displays the system default app set up under the Apps view.
- Use Custom. 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 the Apps view.

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

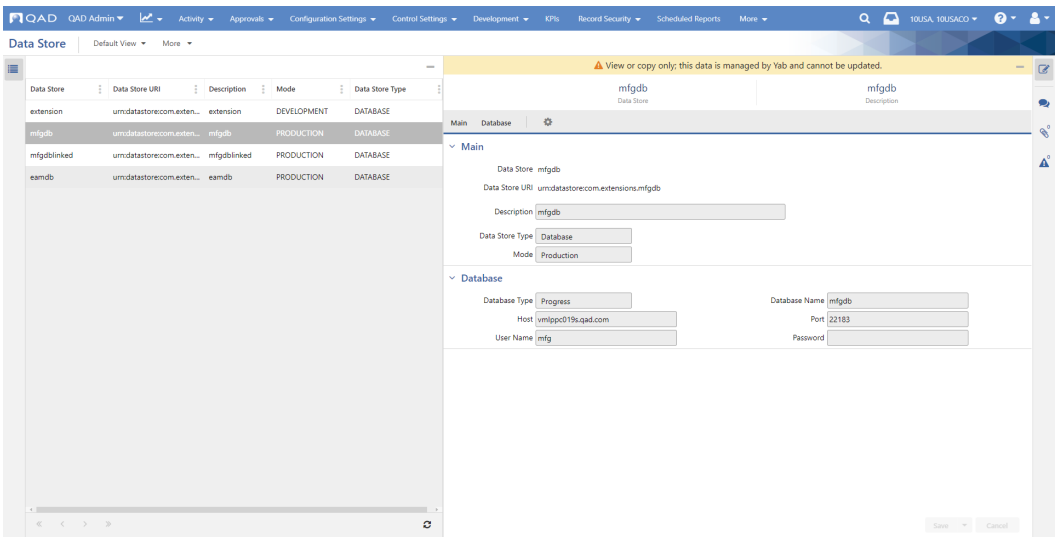
Displays the App URI of the currently active app.

Data Store view

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

Table of Contents:

Child Pages:

Introduction

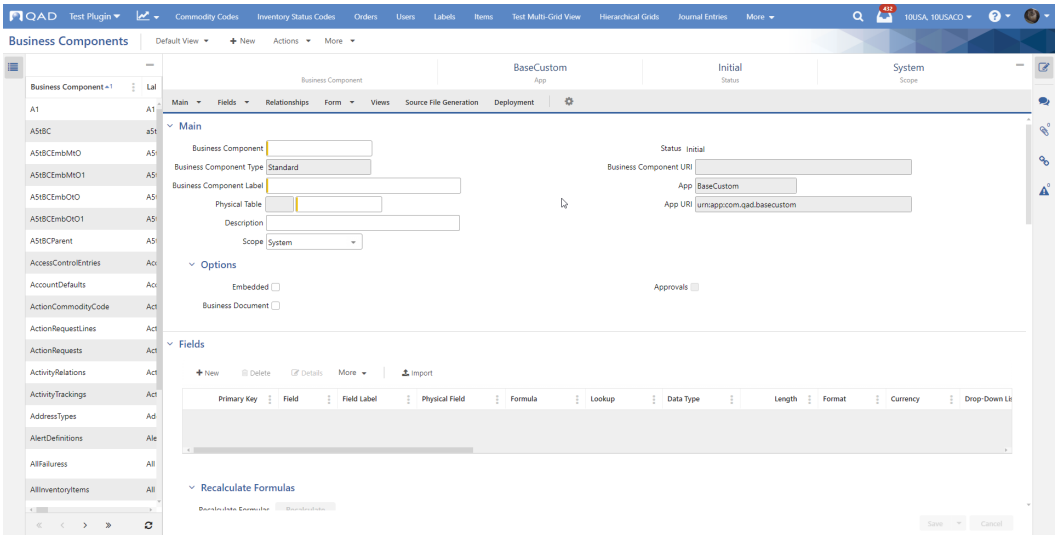
The Business Components view allows a developer to see the list of business components (BCs) in the system (including the ones coded using Progress), and to define new business components dynamically, from scratch. All BCs from all apps are visible, but only the BCs in apps that belong to the environment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

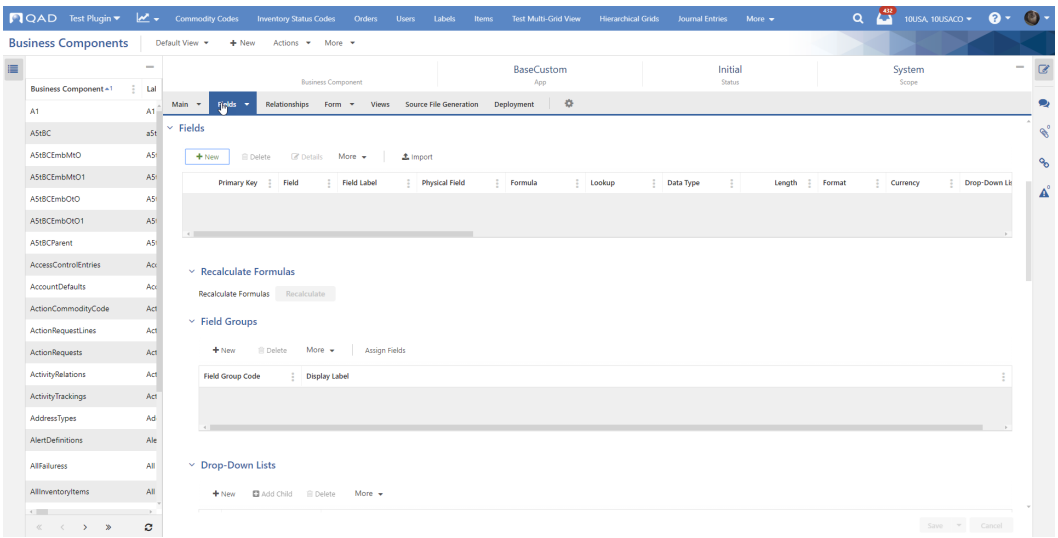
Link: Business Component

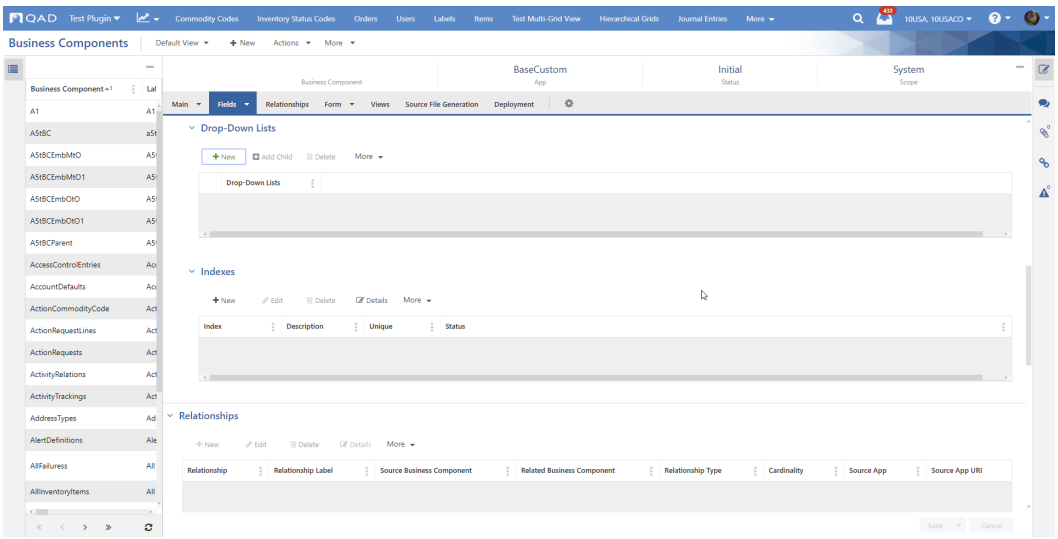
What is a Business Component?

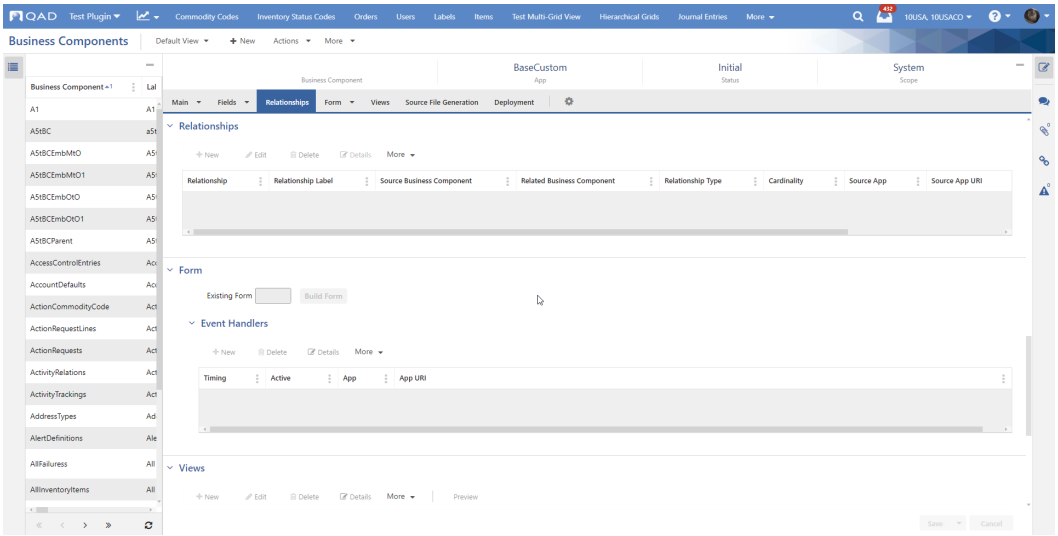
Business Components represent data that functionally belong together, which can be maintained in the system. The data related to BC is of two kinds: metadata describing the BC itself (tables, fields, field groups, etc.) and data of real instances of this entity. The Business Components form displays 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 can 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.

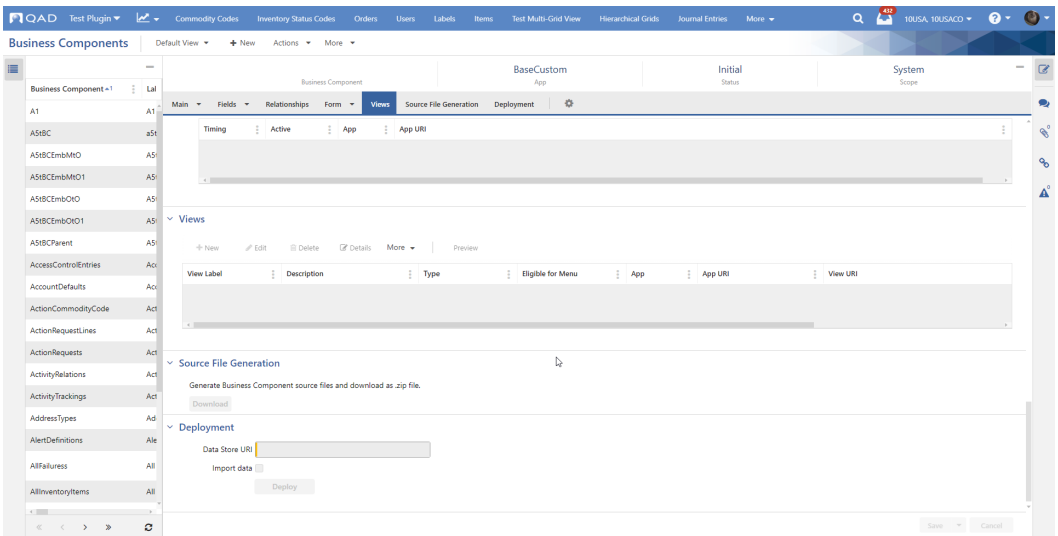
Example Screen Shots


Business Components
Main
Fields
Relationships
Form
Views
Source File Generation
Deployment
Options
Recalculate Formulas


Business Components
Fields
Recalculate Formulas
Field Groups
Drop-Down Lists


Business Components
Drop-Down Lists
Indexes
Relationships





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.

Configuring Panels

The gear icon on the navigation bar opens the Configure Panels dialog where you can control the display of panels and fields on a form. Select a checkbox to display a panel or field; clear a checkbox to hide a panel or field. Fields that do not have labels are indicated by their identifiers within square brackets.

You can use this feature to simplify the form as needed so that you just see the panels and fields you need for a particular task.

You can save these configuration settings as part of a stored view.

In the Configure Panels dialog, panels and fields hidden by business logic are indicated by the "eye-slash" icon. Panels or fields hidden by business logic will not display on the form, even if the display checkbox is selected. Conversely, any panels and fields hidden because the display checkbox was cleared will not display even if the business logic requests that they be displayed.

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, child-parent data relationships.

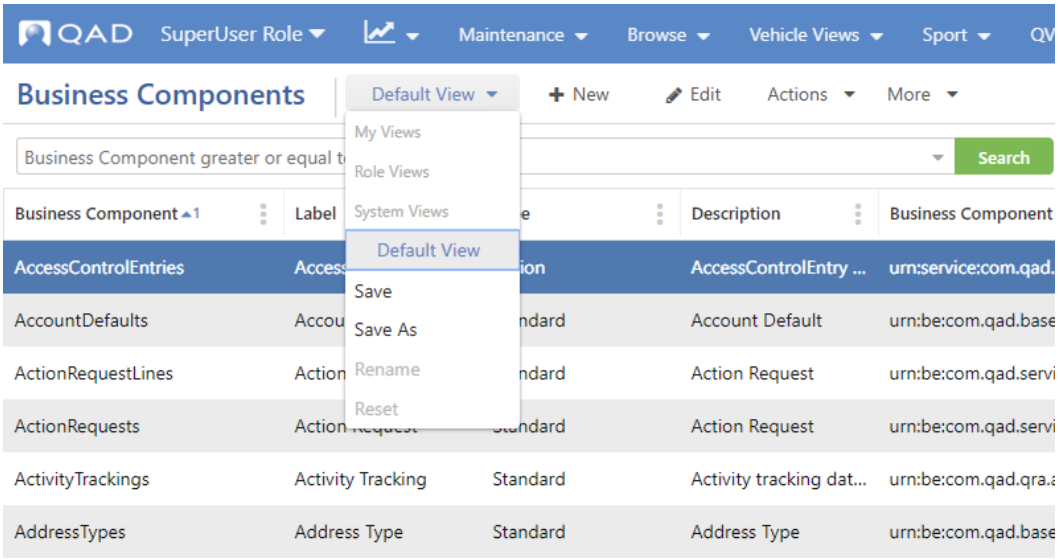
Personalization

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.

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. For more information about stored views, see Stored Views in the online help.



You can perform the following actions from the views menu:

Save

If you make changes to the view and click Save when an original Default View is selected, it becomes a custom Default View. This functionality allows you to customize the Default View by hiding certain fields or including search criteria.

Save As

If you make changes to the view and click Save As, it becomes a stored view. You can rename it to Default View, and it will become a custom Default View as well.

Rename

Change the name of the selected stored view. You cannot rename the Default View.

You can rename a stored view to Default View only if there is no custom Default View yet. In this case, you should always name the Default View in your current language. Note that only administrators can override the Default View.

Reset

Cancel changes you have made to the Default View and set it to the original state. This option appears only when a custom Default View is selected.

Delete

Remove the stored view if you no longer need it. This option appears only when a stored view is selected.

Note that actions related to the custom Default View are available in the com.qad environment namespace only.

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 data types 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 platform business component, Business Component Type is automatically set to Standard and disabled. The creation of a platform 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 that 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. The app registration code is included as a prefix for the business component physical table name. If there is no app registration code, the field is blank. For more information about the app registration codes, see App Registration.

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 My Developer Settings view.

Options panel

Embedded

Indicates whether a business component is either stand-alone or can only be used as a data extension of some other business component, either with a One-to-one relationship or a Many-to-one relationship.

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.

Approvals

Indicates whether approvals are enabled for this business component. 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.

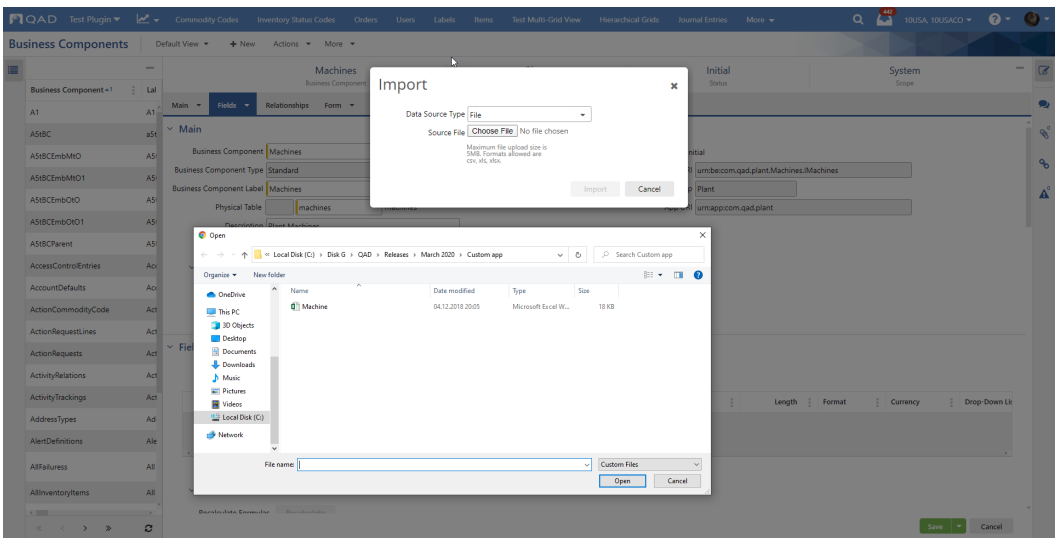
Fields panel

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 the lookup-related business components and typically for use with aggregation functions, and fields from the typical or embedded business component relationships.
- 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.

Note: The Formula definition can be edited after the deployment, but the BC needs to be saved and re-deployed again after making changes.

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.

Field URI

Unique identifier for a field. The Field URI is generated automatically based on the field name and business component. By default, this column is hidden in the grid, but it can be shown by using Configure Panels. The Field URI is added to ensure all annotations in the business component dataset definition are supported by the Business Components screen. (This field is read-only.)

Lookup

Indicates whether a lookup based on the lookup relationship is set up for the field. You can set up a lookup relationship 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.) You can define which browse of the Related BC will be shown in the runtime for the lookup by selecting the Lookup checkbox, and then selecting a specific browse in the Browse field.

Also, there are stored views on lookups. This way you can configure and save your preferred view of the screen. When a lookup is opened, the last used stored view is displayed.

To create a drop-down list based on a lookup, select the Visualize as Drop-Down List checkbox, and then select the value from the Field Used For Label lookup. The values of this field will be displayed in the drop-down list. If you leave the Field Used For Label lookup blank, it is set to the field from Field Mapping > Map To.

Data Type

Select the data type: Character, Currency Amount, 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.

Currency

You can configure rounding of decimal values based on the currency settings. A business component field can have a data type of "Currency Amount", and instead of defining a format, a currency field is selected. Click the lookup icon and select a currency field from the current business component. For this, you should add a field in the business component to represent the currency. For example, add a field called "Currency" (it should have a data type of "Character"). In the runtime, the decimal field will be rounded based on the currency field selected.

To configure Statutory Currency, add a field with a data type of "Currency Amount", and then type STATUTORYCURRENCY (all caps) in the Currency field (don't select anything from the lookup). In the runtime, the currency field will show the correct number of decimals according to the selected domain. To configure Base Currency, follow the same steps as for Statutory Currency, but type BASECURRENCY (all caps) in the Currency field instead.

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 in the runtime.

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.

Description Field

Identifies a field as a description field, meaning it has an Associated Code field. The Description Field enables the Associated Code field and is just a helper field for Associated Code. These fields are added to ensure all annotations in the business component dataset definition are supported by the Business Components screen.

Associated Code

Description Fields are linked to their Associated Code fields. Field security then will hide the Description Field in the resource tree, but its access permissions will be inherited from the Associated Code field. The Associated Code field is enabled only when the Description Field is selected. It contains a lookup to the Fields browse to select a field. The Associated Code field is added to ensure all annotations in the business component dataset definition are supported by the Business Components screen.

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.

Hidden from Activity Tracking

Indicates whether the field is enabled for activity tracking.

Hidden from UI

Indicates whether the field is hidden from the views, form, drill downs, alerts, lookup definitions, runtime, activity tracking, design layout, etc.

Sortable

Indicates whether the field is sortable. It is always set to Yes (selected) for primary fields and cannot be changed.

User Defined

Indicates whether the field is user-defined. (This field is always read-only.)

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 relationship to bind it to the business component it extends.

Deployed

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

Discriminator

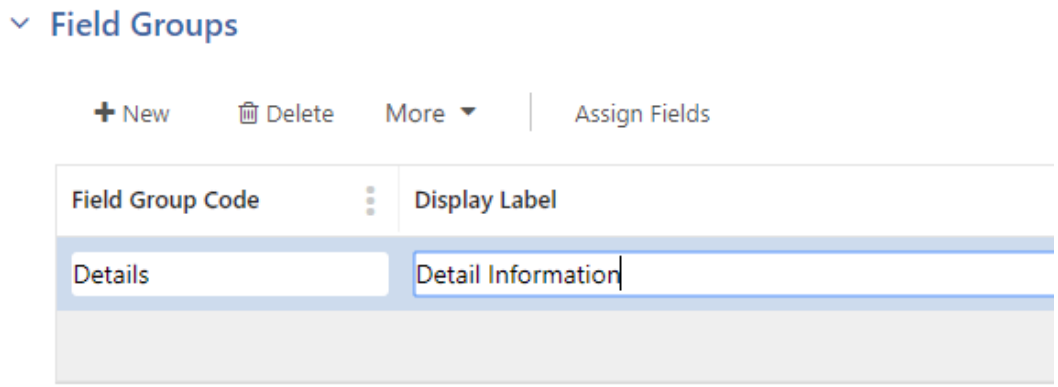
Indicates whether the field will be used as a "discriminator" field. A "discriminator" field is a sort of helper field used only for embedded BCs to provide the facility of DEO BCs extending several parent base BCs via 1-1 relationship. This field must be a part of primary key and will not be visible 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.

Recalculate Formulas panel

Adding a new Formula field to already deployed BC will undeploy this BC and you will need to deploy it manually again. If there are any existing records for this BC when a new Formula field is added, they should be recalculated by CalculationEngine (to have the value of the newly introduced Formula field stored in the database for those records). This is done automatically when you re-deploy that BC. If you add this newly introduced Formula field to the Browse, the values for this field might not appear there for some time (until the calculations are finished). If, for some reason, you want to force Formula fields recalculation, click the **Recalculate** button.

Field Groups panel

Field Groups are a selection of fields that can be associated with access rights of a role or roles.



New

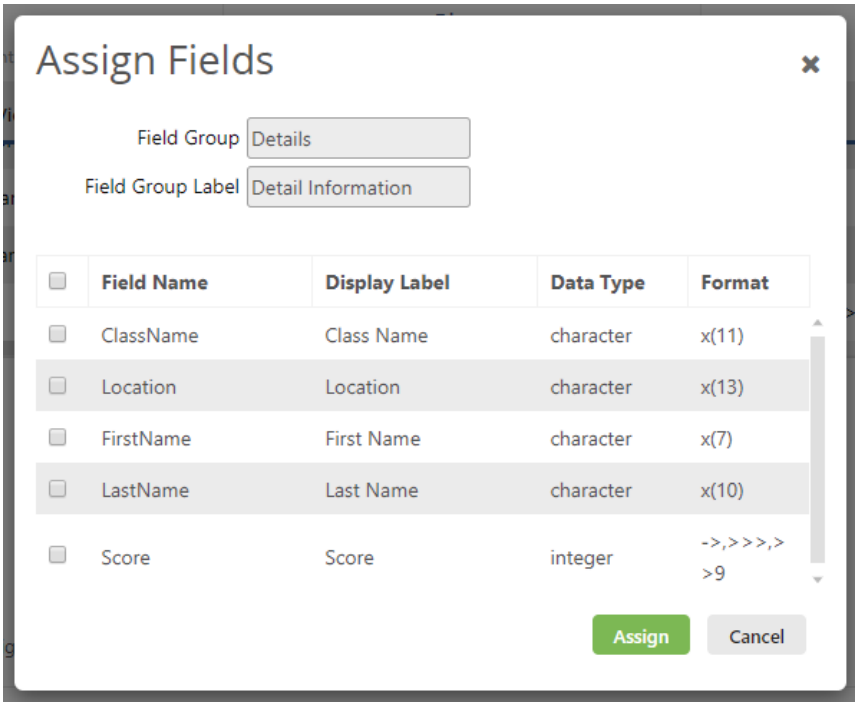
Create a new field group.

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

Define a business component field as a Drop Down and set up drill-down lists for the business component, which can then be assigned to a field.

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.

Indexes panel

You can set up extra indexes for platform-generated business components (not coded business components). The Indexes panel is hidden for coded business components.

A secondary index enables rapid sorting and grouping by additional columns.

New

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

Edit

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

Delete

Delete an existing index.

Details

This opens the detail form view for entering further details.

Index

Enter a name for the index.

Note: A primary index (called idx_PK) is automatically generated when a business component is saved. When a primary index is automatically generated, a description of "Primary Index" is included (but is still editable).

Before the business component is deployed, the status is "Not Active - Deploy Required", and the index Description, Unique, and Order can be changed. After the deployment, the primary key index cannot be modified.

Description

Add a description for the index.

Unique

Specify whether this index is unique. When you select this checkbox, the combination of field values is unique in the system, and it does not contain any two rows with the same values of fields specified in the index.

Status

This field is generated automatically (on the Java side) based on two parameters: if it is active and if a BC has a table in the database. You can create a new index before and after the deployment. After the deployment, you cannot edit the existing indexes. All existing indexes related to a particular B table are activated during the first deployment. If you create the index after the first deployment, it is created automatically on the DB level as inactive. To activate such indexes, you will receive a notification and will need to run the following YAB command:

yab stop database-<DB_NAME>-index-rebuild -inactiveindexes start.

Status Additional Description

- "Not Active - Deploy Required" — The BC is not yet deployed. The table and indexes are not created yet in the database, only metadata is created for the index, the index is not created on the DB level yet. It will be created during the BC deployment. The index can be deleted in this status.
- "Not Active - YAB Command Required" — The BC is already deployed. After the BC deployment, the index is created on the UI and saved. When saving the BC, it is created on the DB level with a flag PRO_ACTIVE 'n', this is the stage where you need to run the YAB command (yab stop database-<DB_NAME>-index-rebuild -inactiveindexes start) to activate the index in the database.
- "Active" — The index is active. The BC is deployed, and the index is created in the database and is activated (through the BC deployment or by running the YAB command).

Fields panel

Select fields for the index from the lookup.

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 relationship. This opens the detail form view for entering further details.

Edit

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

Delete

Delete an existing BC relationship.

Details

This opens the detail form view for entering further details.

Relationship

The code of relationship between this business component and the other business component.

Relationship Label

The label for the business component relationship.

Source Business Component

Specifies the business component on which a relationship was created.

Related Business Component

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

Relationship Type

It is always set to Child and cannot be changed. For more information about relationships, see Business Component Relationship.

Cardinality

Specifies the cardinality of the relationship: One-to-one or Many-to-one.

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 One-to-one 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 "One" 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. Note that you need to create the business component relationship from a child if you want to have a business document with parent and child.
- Enable a business document under Main > Options panel.

Note: Excel integration (import and export) is available for all business components (including embedded 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 that 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 Label

A display label for the view.

Description

A short description 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 business components can have the following cardinality: One-to-one and Many-to-one. Many-to-many relationships between business components are not supported.

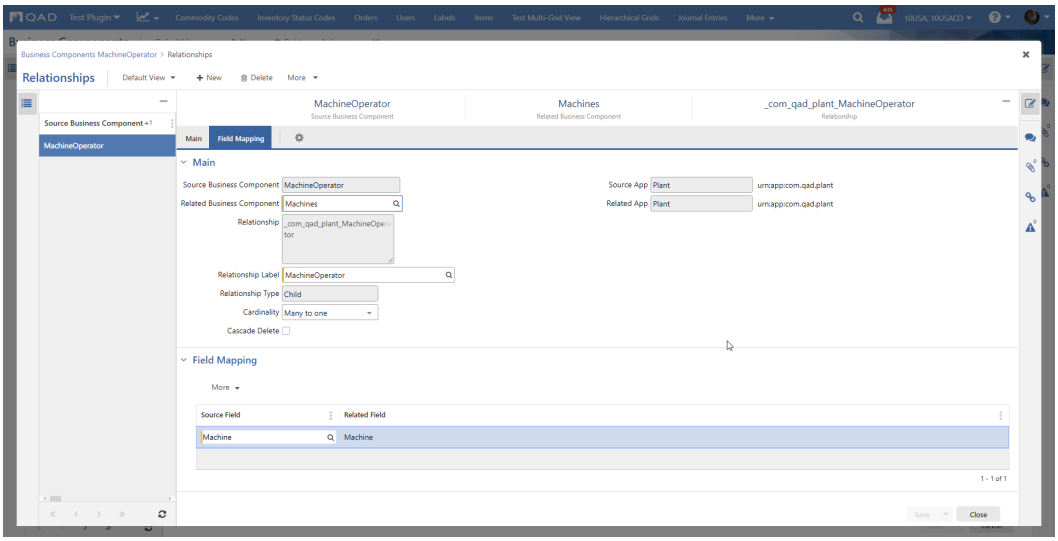
Previously, One-to-many relationships were also possible. However, to avoid creating not unique relationships between two business components both from Child and Parent side, there was a decision to allow creating/editing relationships only from the Child side. Thus, starting with the September 2019 release, only One-to-one and Many-to-one cardinalities are possible. If you still want to create One-to-many (parent-child) relationships, you can do that by creating Many-to-one (child-parent) relationships from the Child side.

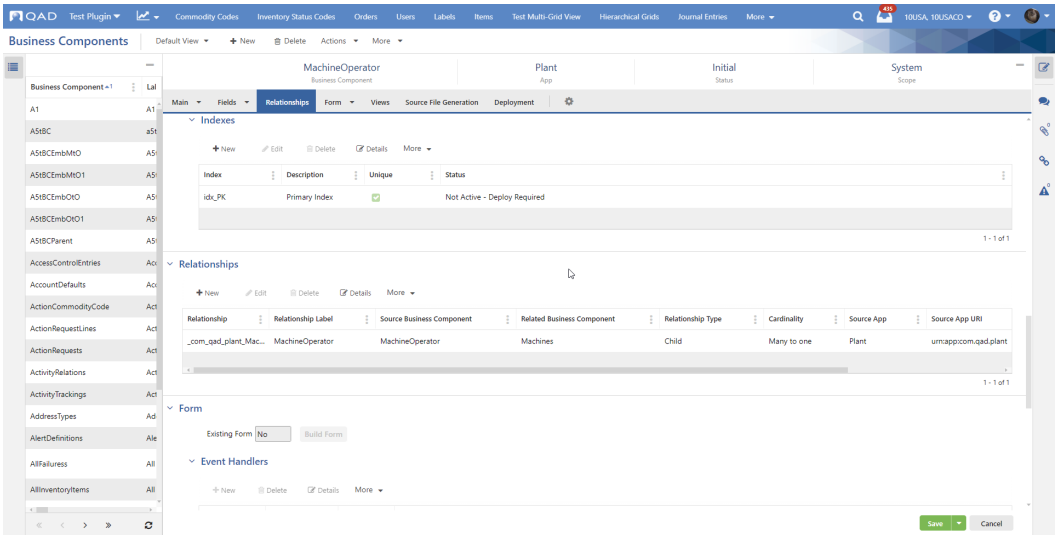
Also, the system only allows creating unique relationships between a pair of business components. You will receive a duplicate error message if you try to create two relationships that are equal.

As with relationships between tables in relational databases, business component relationships are based on primary keys. To be able to create One-to-one relationships between two business components, they must have the same number and type of fields for their primary keys. For Many-to-one relationships, the "Many" 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 business component on the "One" side.

Screen Shots

You can open Relationships by 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. Child-parent relationships cannot be edited; they can only be deleted and then recreated again. If the last child-parent 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 relationship 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 Source Business Component, but can be changed. It also includes the full name of the app when saved.

Relationship Label

Displays the relationship label. This field is automatically populated based on the Source Business Component, but can be changed.

Relationship Type

It is always set to Child and cannot be changed. For more information about relationships, see Business Component Relationship.

Cardinality

Specifies the cardinality of the relationship: One-to-one or Many-to-one.

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 One-to-one 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 "One" 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.

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 the "One" side of the relationship, and the developer cannot edit this column in the grid. The fields from the "Many" 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.

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 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. You can sort and filter by Formula fields in Browses.

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

You should think thoroughly of the Data Type of the Formula field you are going to create. The field's Data Type must match the result of the formula definition expression, otherwise, the CalculationEngine will fail to store the Formula field value in the database (or will store incorrectly) and there is no way to change the Data Type of the Formula field after the BC is deployed.

Adding a new Formula field to already deployed BC will undeploy this BC and you will need to deploy it manually again. If there are any existing records for this BC when a new Formula field is added, they should be recalculated by CalculationEngine (to have the value of the newly introduced Formula field stored in the database for those records). This is done automatically when you redeploy that BC. If you add this newly introduced Formula field to the Browse, the values for this field might not appear there for some time (until the calculations are finished). If, for some reason, you want to force Formula fields recalculation, click the **Recalculate** button on the Fields > Recalculate Formulas panel in the Business Components screen.

If you change the definition of a formula field, the BC is also undeployed and you will need to redeploy it manually.

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 the related BC (lookup, typical, and embedded relationships): relationName.fieldName prepended with the App URI

The CalculationEngine can calculate formulas, which have up to six levels of relationships. You can configure this setting using the qad-qracore.formula.relation.maxLevel property. The default value is 3 levels due to performance issues.

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

The 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',

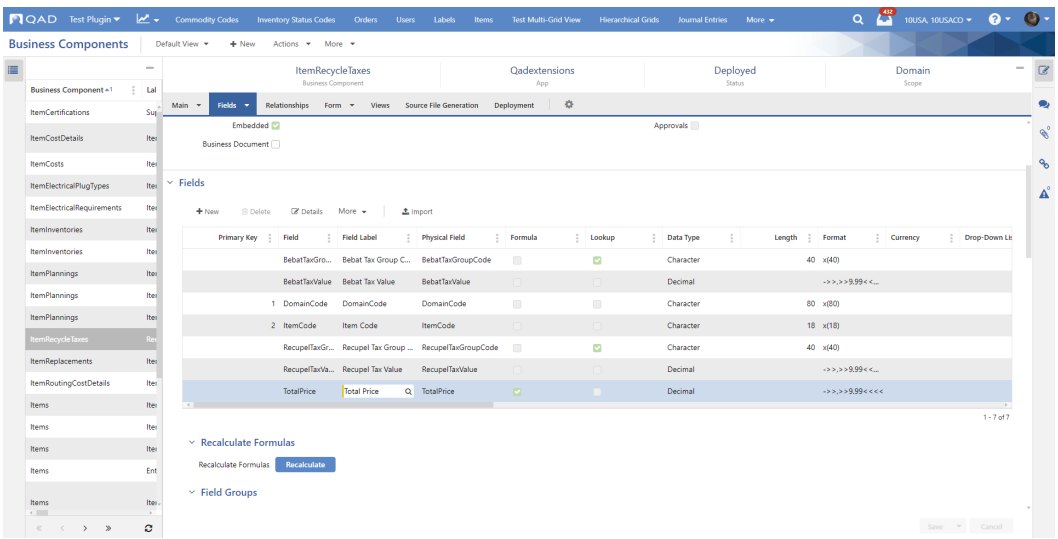
'IMAGINARY', 'IMARGUMENT', 'IMCONJUGATE', 'IMCOS', 'IMCOSH', 'IMCOT', 'IMCSC', 'IMCSCH', 'IMDIV', 'IMEXP', 'IMLN', 'IMLOG10', 'IMLOG2', 'IMPOWER', 'IMPRODUCT', 'IMREAL', 'IMSEC', 'IMSECH', 'IMSIN', 'IMSINH', 'IMSQRT', 'IMSUB', 'IMSUM', 'IMTAN', 'INT', 'INTERCEPT', 'INTERVAL', 'IPMT', 'IRR', 'ISBINARY', '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 "Many-to-one" child-parent BC relationship defined), but for these fields only the following operators are applicable:

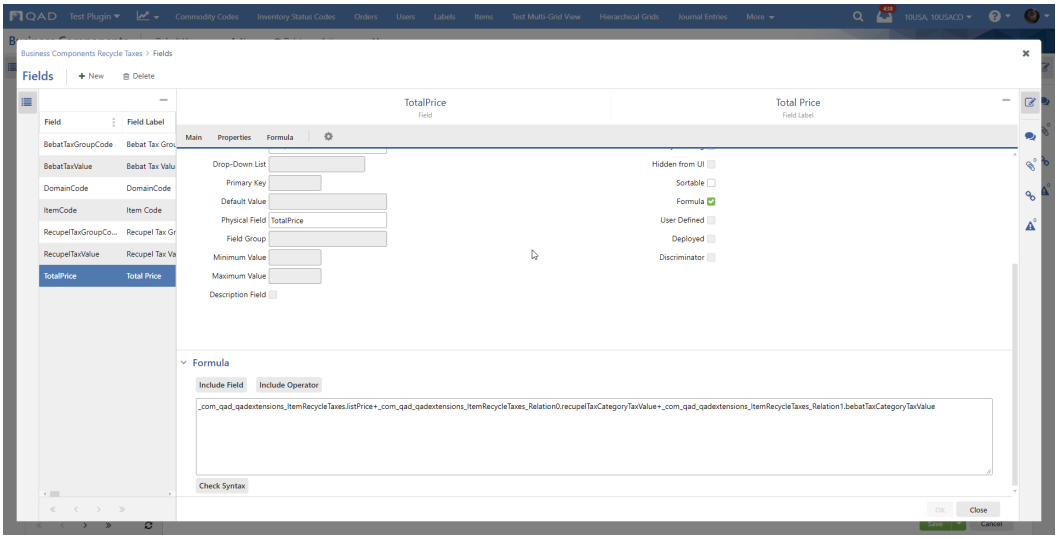
- SUM
- MIN
- MAX
- COUNT
- AVERAGE

Example Screen Shots

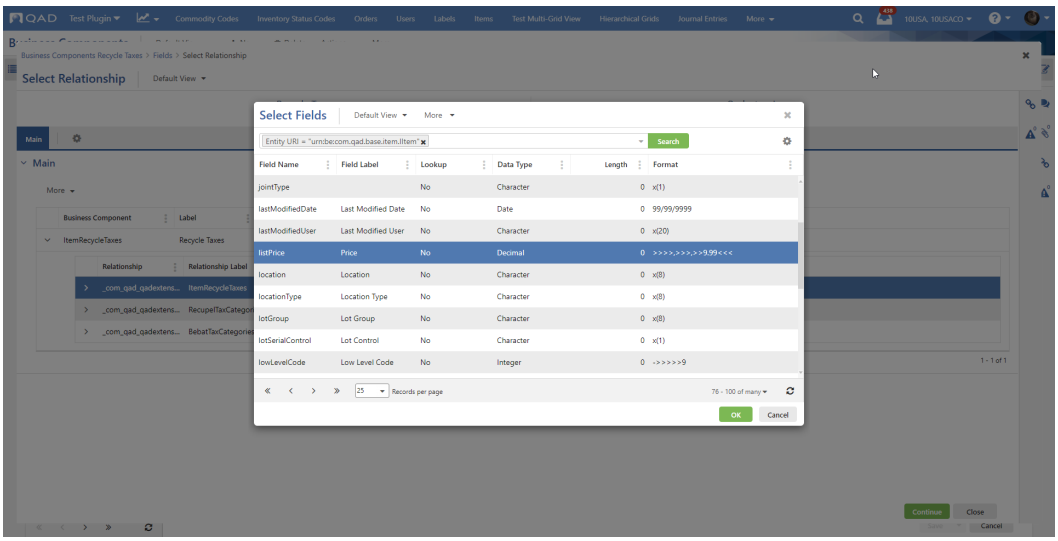
Business Components view > **Fields panel**



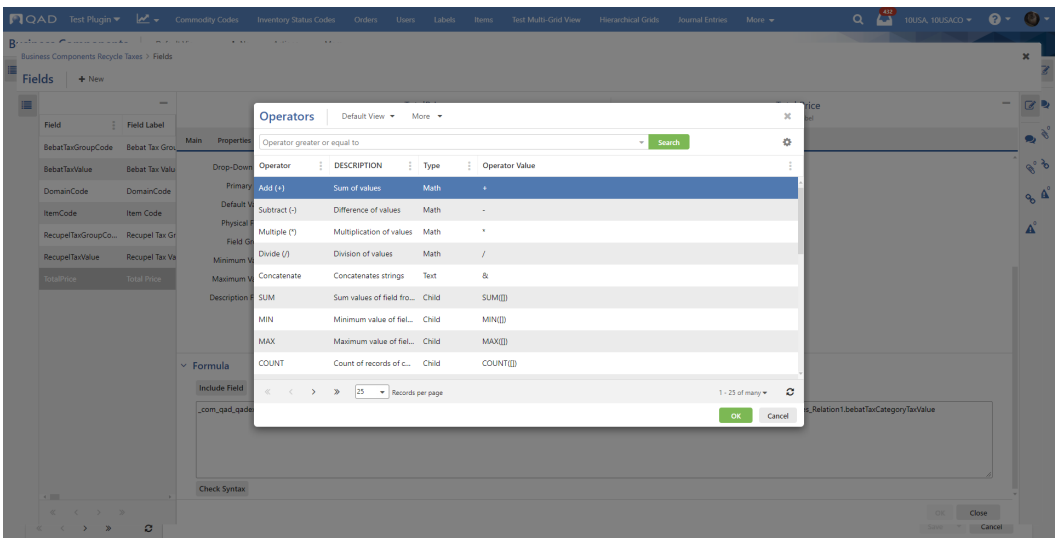
Business Components view > Fields panel > Details button > **Formula panel**



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 Select Relationship > Select Fields dialog, 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 dialog, 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.

Select Relationship pop-up window (Include Field)

When you click the Include Field button, the Select Relationship pop-up window opens. When you expend the relationship, it will show the relationships up to three levels by default. You can select the relationship you need, click Continue, and then in the Select Fields dialog, add columns/fields from this relationship.

Select Fields dialog

Field Name

The name of the formula field.

Field Label

The display label of the formula field.

Additionally, the Lookup, Data Type, Length, and Format columns are displayed for unique identification of the field.

Operators dialog (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

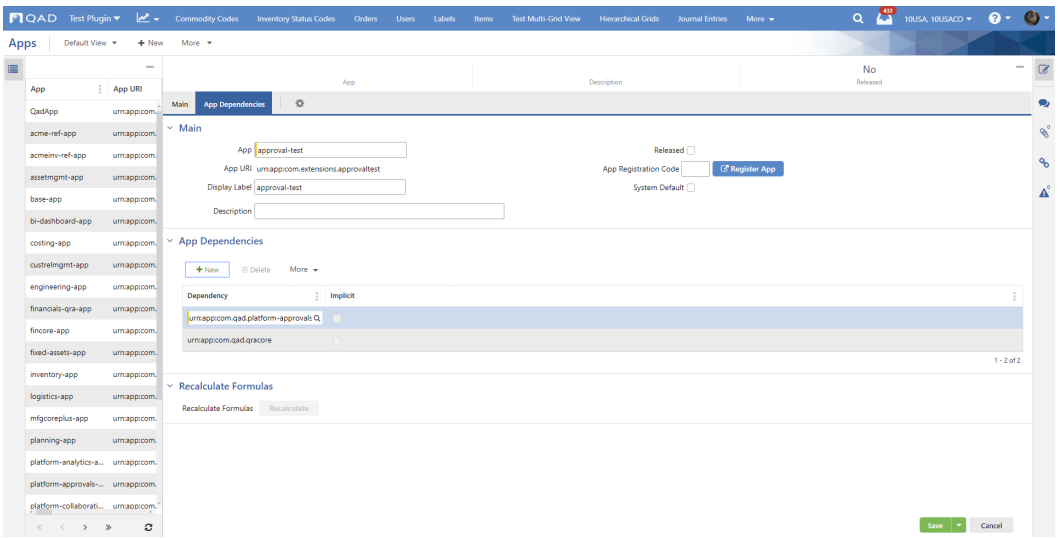
The steps for creating an approval flow for a platform business component are as follows.

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.

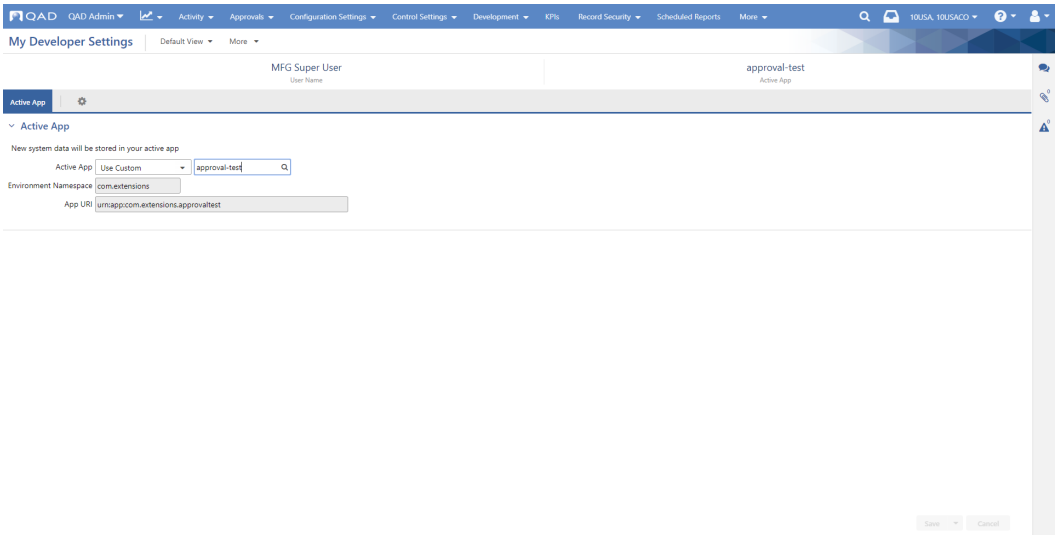
Our new app should have a dependency on the standard app **platform-approvals-app** and on qracore. Note that the dependency on qracore will automatically be added when you save the app with the other dependencies in place.

1. Launch **Apps** from the menu search, 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**.



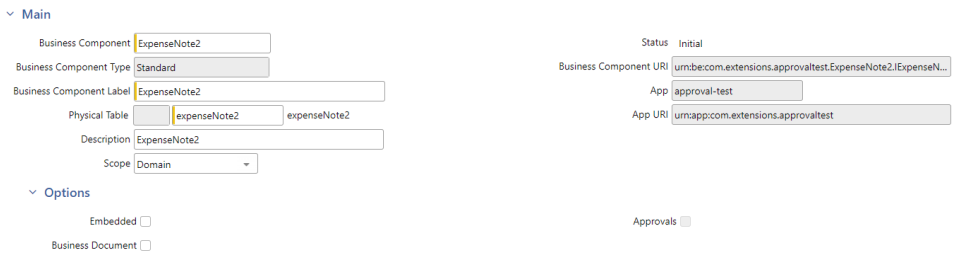
Step 2: Set the new app as Active for yourself

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

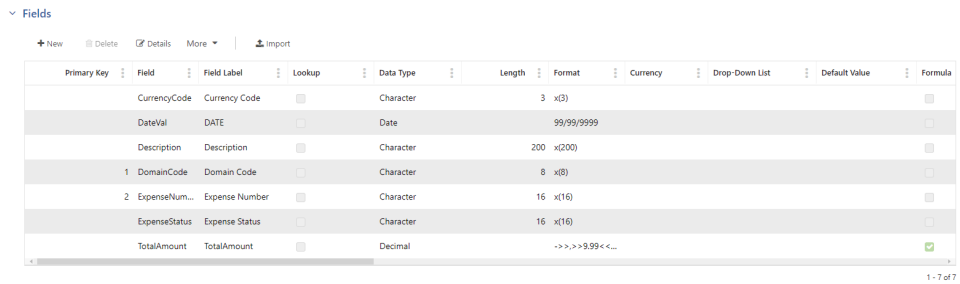


Step 3: Create and deploy a platform business component

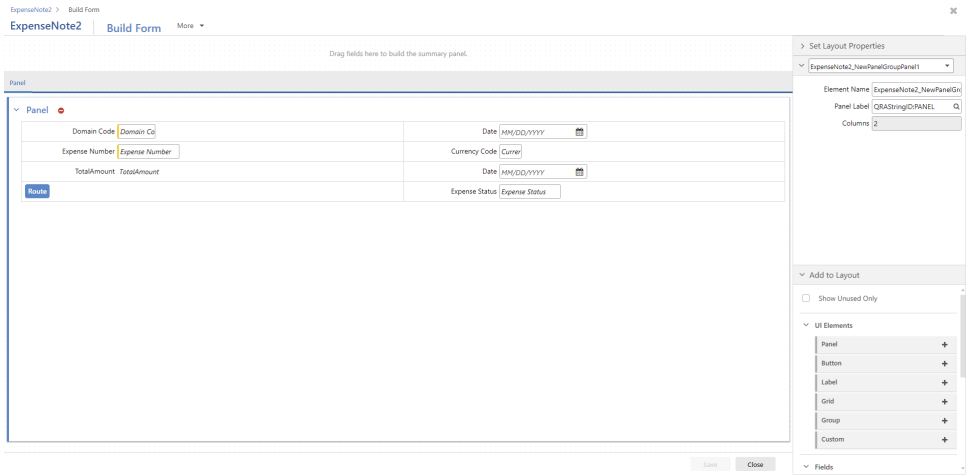
1. Launch **Business Components** from the menu search, and then create a new business component.
2. Fill in the **Main** panel as follows.



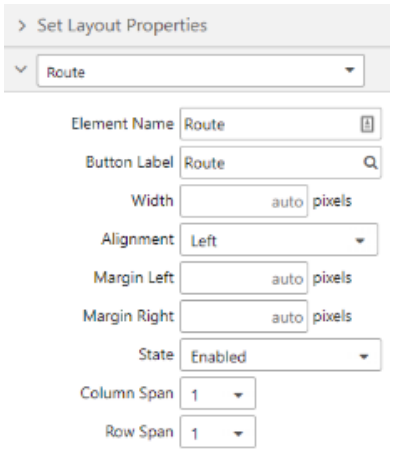
3. On the **Fields** panel, fill in the Fields as listed on the screen shot below. There should be a field in the BC to represent the "Status", like in this example the **ExpenseStatus** field. Note that you also need to add DomainCode as a part of a **Primary Key** field because the **Scope** was selected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



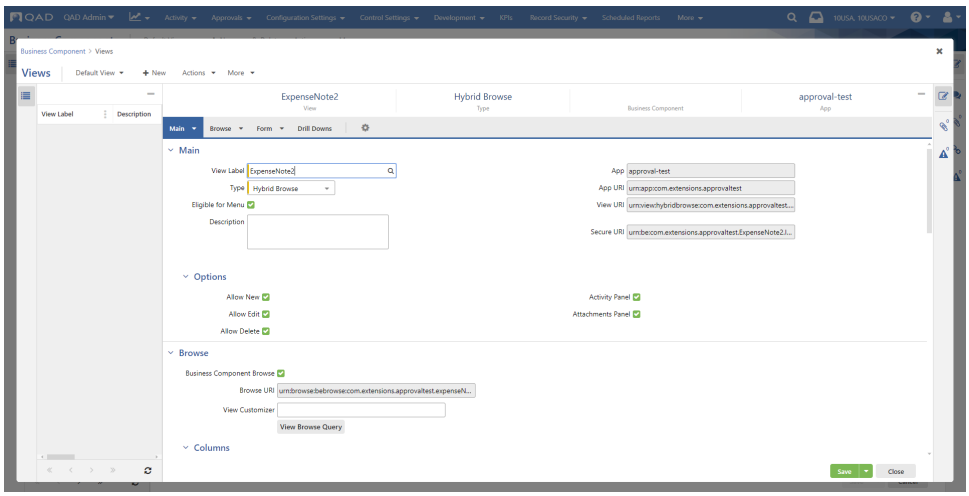
4. **Save**.
5. Navigate to the **Form** panel, and then click the **Build Form** button. In the **Build Form** pop-up window, compose a Form: drag-and-drop all fields on the Panel, drag-and-drop the UI element "Button" (name it "Route") on the Panel, and then click **Save** and **Close**.

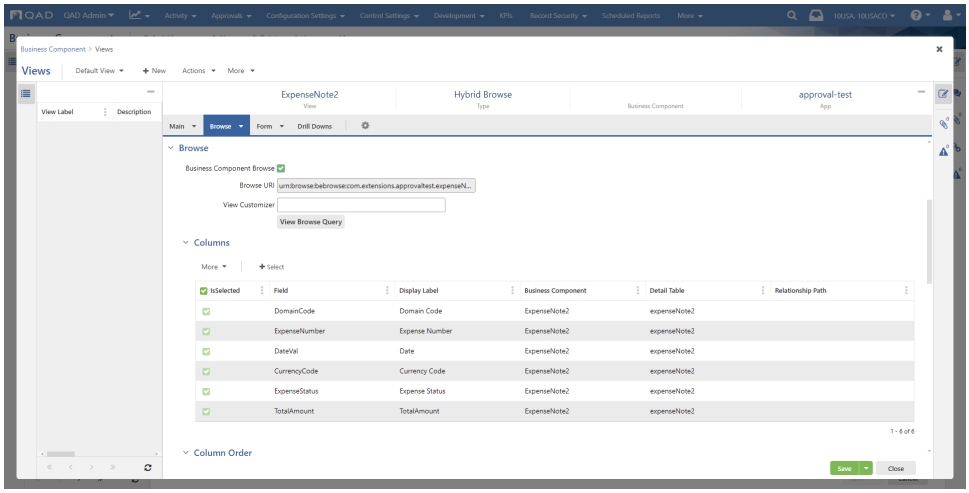


Note: The Route button should have the following properties: Element Name and Button Label as "Route". The properties can be different but it should be considered in the Event Handler that handles this button.

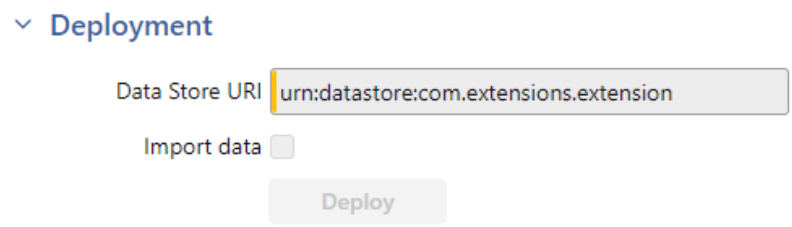


6. Navigate to the **Views** panel, and then click the **New** grid button. In the **Views** pop-up window, compose a Hybrid Browse, and then click **Save** and **Close**.

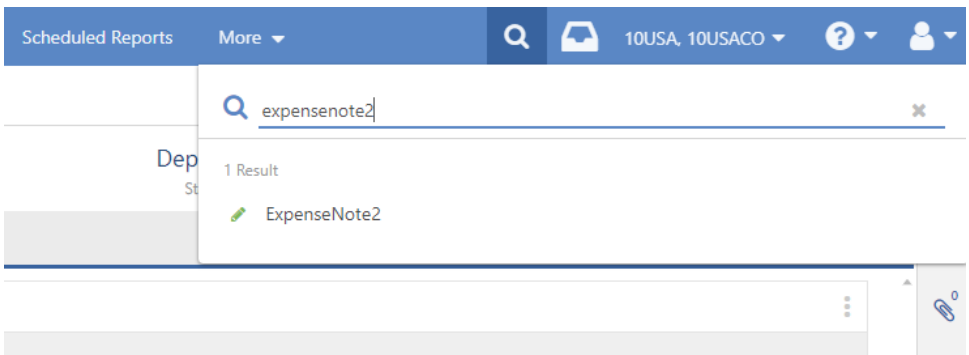




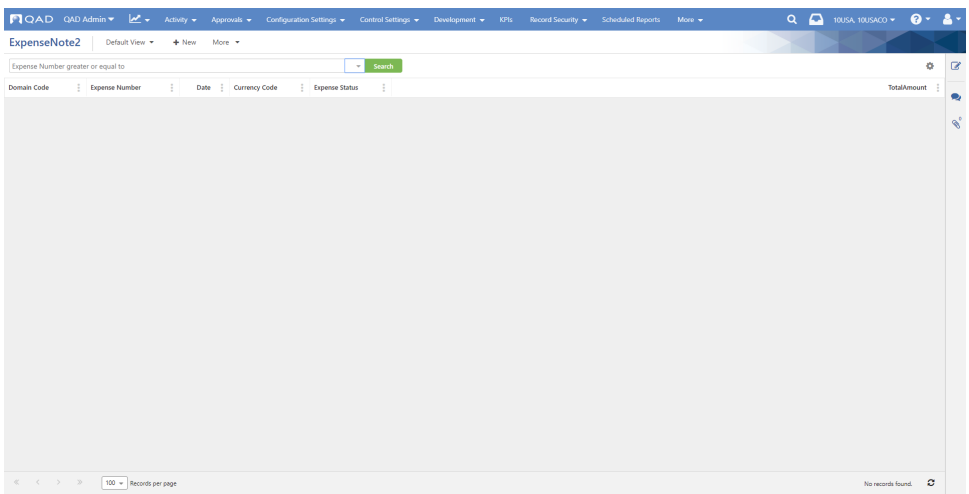
7. Deploy the business component: locate the **Deployment** panel, select a data store, and then click **Deploy**.



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:

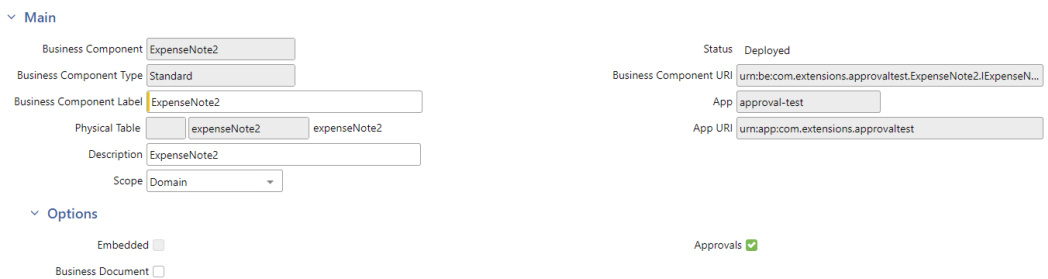
1. Navigate to the created business component (ExpenseNote2) > **Form** panel > **Event Handlers** panel, and then click the **New** grid button.
2. In the **Event Handlers** pop-up window, set the handler as "Active", "Primary".
3. Add a TypeScript code, click the **Compile** button, and then **Save**.

Typescript Code

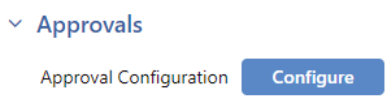
```
module com.extensions.approvaltest.EventHandler.ExpenseNote2.ComExtensionsApprovaltest.
Maint_PRIMARY {
    "use strict"

    import QraViewTSHandlerWithViewFormTSHandler = Qad.QraView.TSHandler.
QraViewTSHandlerWithViewFormTSHandler;
    import QraViewFormTSHandlerV2 = Qad.QraView.TSHandler.QraViewFormTSHandlerV2;
    import IViewField = Qad.QraView.TSHandler.IViewField;
    import DTO = com.extensions.approvaltest.EventHandler.ExpenseNote2.DTO;
    import Constants = com.extensions.approvaltest.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 {
                        // Start the approval process. Note that the first parameter is the name
of the button you added on the form with the form builder.
                this.startApprovalProcess("Route",this.$scope, eventData);
        }
    }
}
```

Step 5: Create an approval for a business component

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

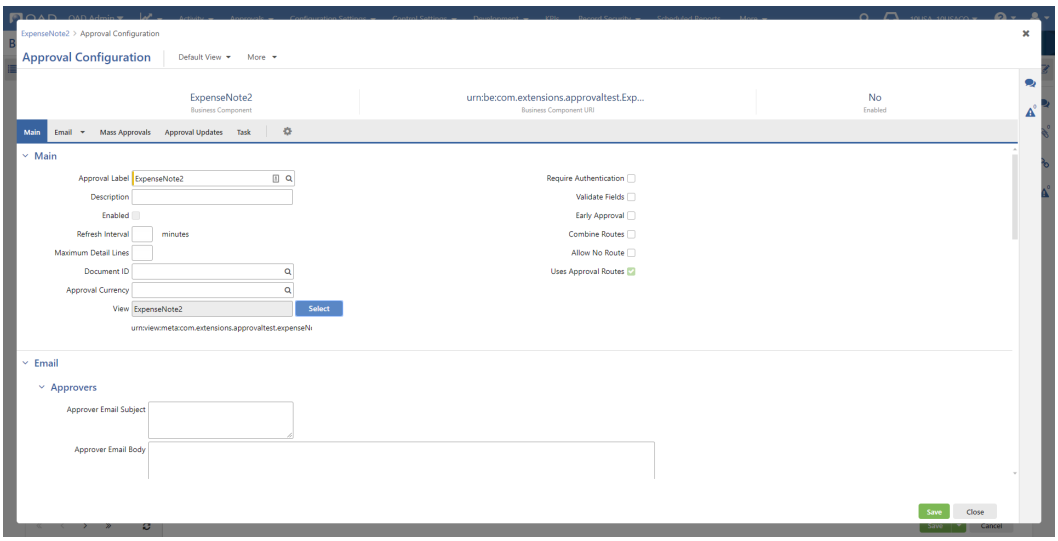


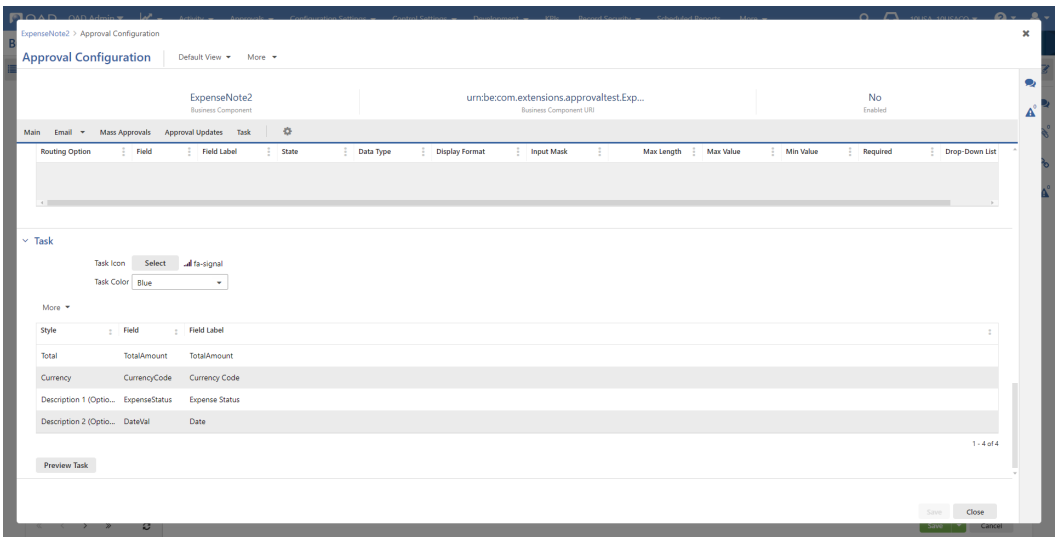
Now the additional **Approvals** panel appears at the bottom of the page. The **Configure** button allows you to configure approval settings for the current business component.



Step 6: Approval configuration

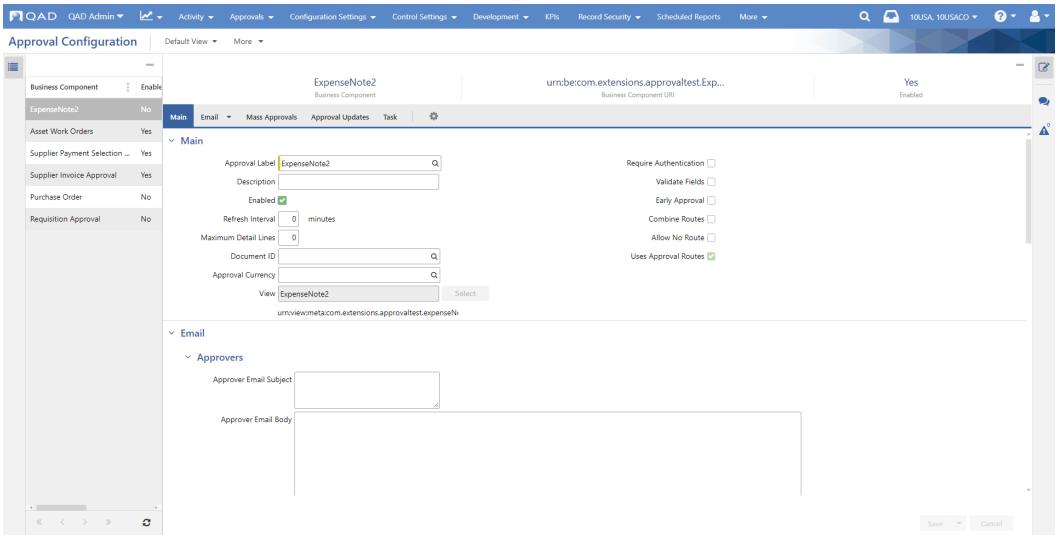
1. On the **Approvals** panel, click the **Configure** button, and then in the **Approval Configuration** pop-up window specify the configuration in all panels.
2. Optionally, you can configure the **Document ID** field to appear as a hyperlink in the Inbox. To configure this functionality, click the **Document ID** lookup, and then select the field that determines the document ID of an approval. Note that it should be a Primary Key field. It is useful in the following cases:
    - When approvers receive a task in their Inbox, they can click the hyperlink and will be redirected to the Future Approval browse to take any action (Approve/Deny).
    - After the task is Approved or Denied, users will receive a notification in their Inbox. They can click the hyperlink and will be redirected to the Purchase Order browse, for example, to view all the details.
3. Save the configuration.

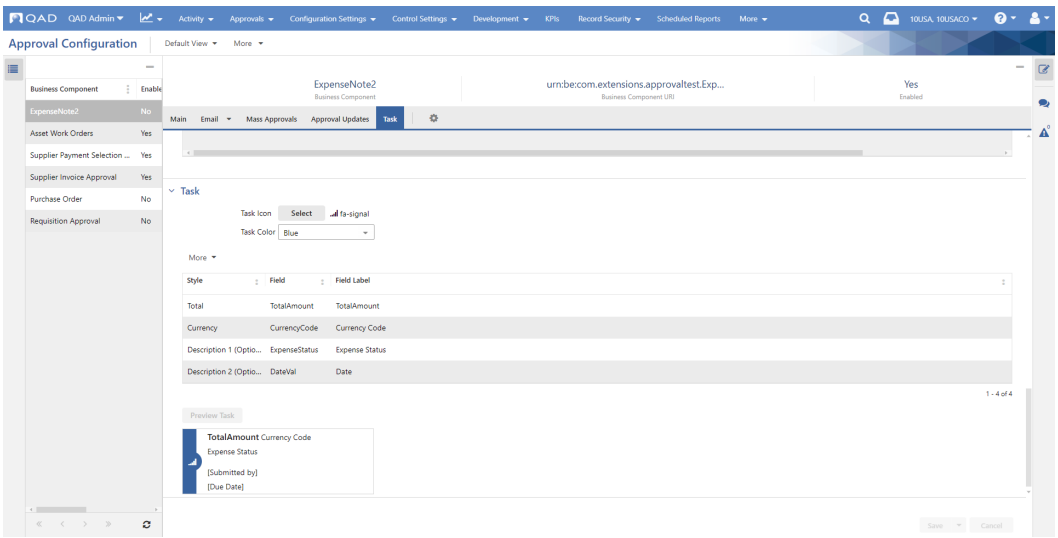




Step 7: Enable approval configuration

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



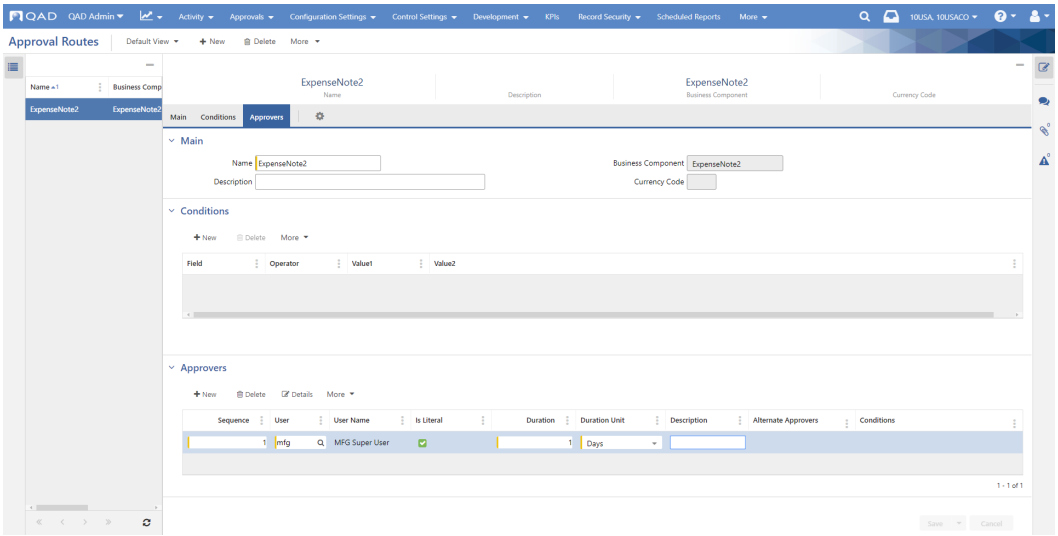


The **Preview Task** button allows you to preview the task according to the selected properties.

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

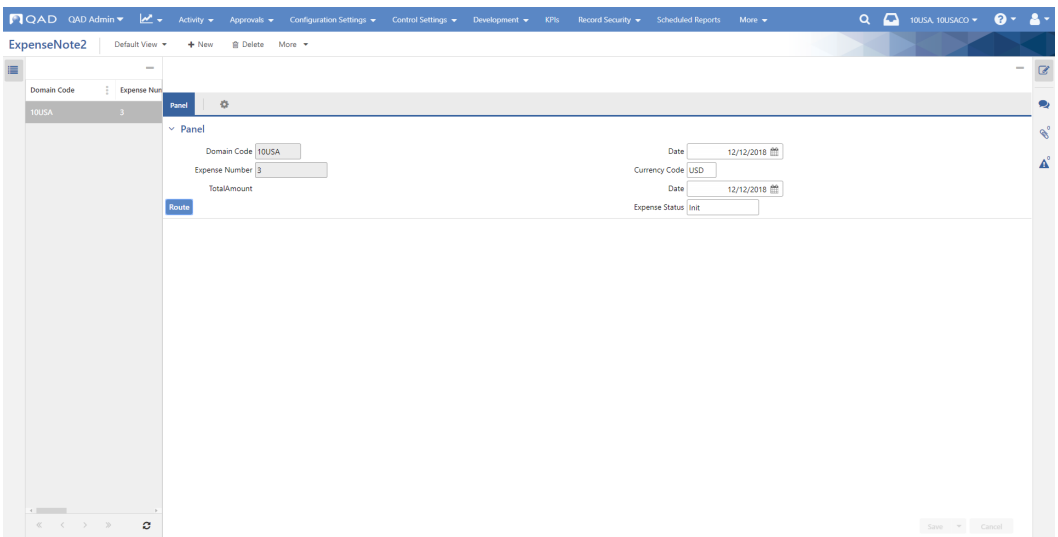
Step 8: Create approval routes

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**.



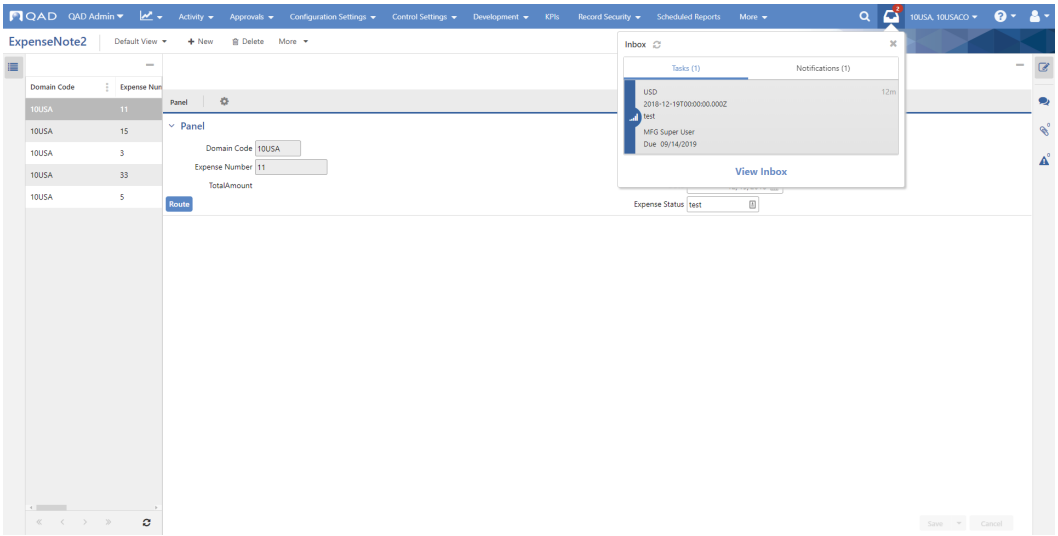
Step 9: Route action to an approver

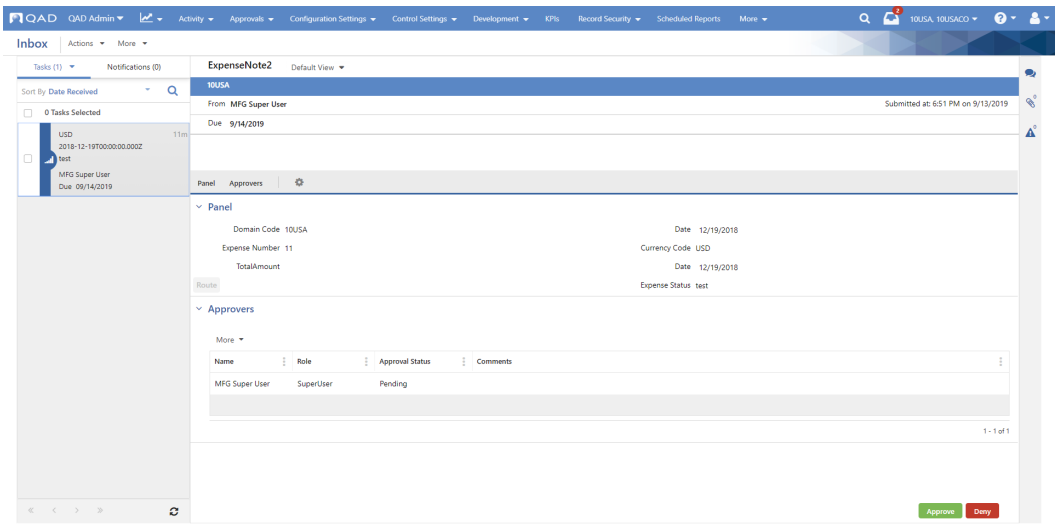
1. Open the business component in the runtime from the menu search, and then click the **New** toolbar button.
2. Create and save a new record.
3. Click the **Route** button. This will create a new task for an approver. (It can take some time, from seconds to minutes).



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.

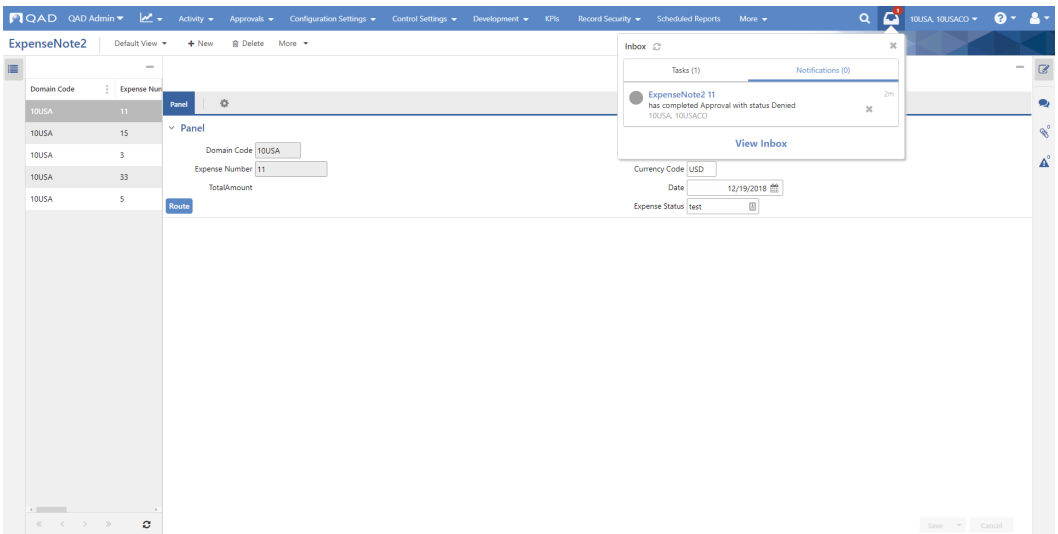




Approvers can approve or deny the task

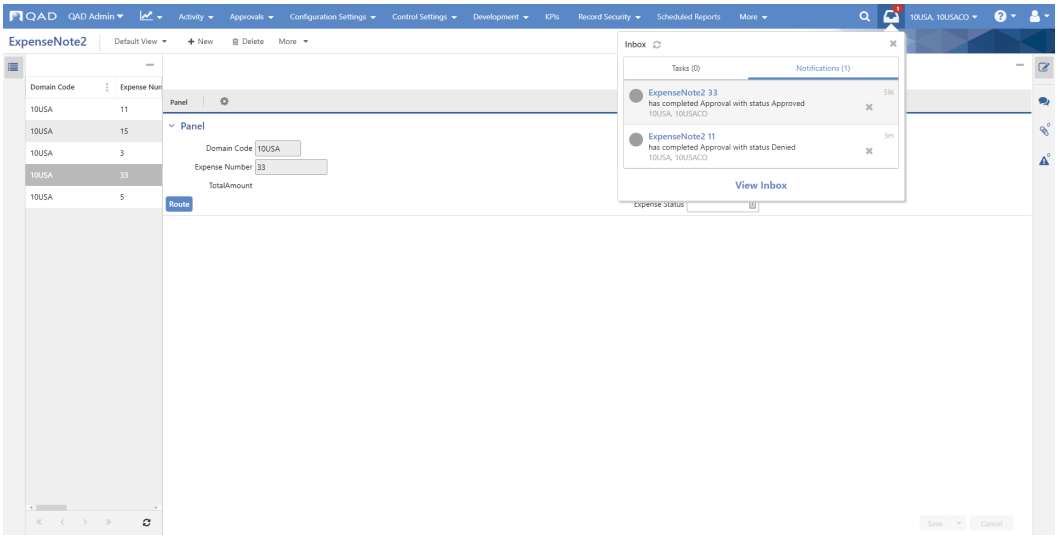
A. Task denied

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.



Step 11: [Optional] Add customization to Approvals

The current step is optional and describes the instructions on how additional customization can be added to Approvals.

A. 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 -appuri:urn:app:com.extensions.approval-test -dir:/dr01/qadapps/systest/approval

```
                    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)
```

B. 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.approval.dir=/dr01/qadapps/systest/approval

```
#----------------------------------------------------------------------
# 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.approval.dir=/dr01/qadapps/systest/approval
```

C. Write customization. Approved records will display the APPROVED status in the Expense Status field.

1. In your yab environment, go to the folder where your customizations will be created. In our case here, that is approval/src/impl/com/extensions/approval-test.
2. All data extensions are handled by **ApprovalVirtualEntity** so we will have to customize this class.
3. Create a subfolder for the customization you are going to do. Since we are going to customize ApprovalVirtualEntity, we create a subfolder approval/entity.
4. In the folder approval/src/impl/com/extensions/approval-test/approval/entity write the custom code. In our case, in ApprovalVirtualEntityCust.cls.

ApprovalVirtualEntityCust.cls

```
/*
 * com.qad.approvals.entity.ApprovalVirtualEntity - Approval Virtual BE Implementation class
 *
 * Copyright 1986 QAD Inc. All rights reserved.
 *
 * $Id: $
 */

/* using com.qad.approvals.entityv3.IGenericApprovalEntityV3. */
using com.qad.approvals.entityv3.IApprovalVirtualEntity.
using com.qad.approvals.GenericApprovalConstants.
using com.qad.approvals.GenericApprovalServices.
using com.qad.approvals.IGenericApproval.
using com.qad.approvals.entityv2.IGenericApprovalEntityV2.
using com.qad.approvals.config.IGenericApprovalConfiguration.
using com.qad.qra.resource.ResourceIdentityUtils.
using com.qad.qra.resource.IResourceResolver.
using com.qad.qra.transaction.ITransactionManager.
/* using com.qad.qra.businessentity.IEntityFetchService. */
using com.qad.qra.QraServices.
using com.qad.lang.IList.
using com.qad.qra.core.GlobalServiceLocator.
using com.qad.qra.QraError.
using com.qad.lang.Message.
using com.qad.lang.Properties.
using com.qad.qra.logging.ILogger.
using com.qad.qra.config.QraConfig.

/* using com.qad.base.currency.ICurrency. */
/* using com.qad.base.currency.IExchangeRate. */

using com.qad.qra.be.IVirtualBusinessEntity.
using com.qad.qra.extensible.IInterfaceDatasets.
using com.qad.qra.metadata.IEntityMapping.
using com.qad.qra.QraMessageKey.

routine-level on error undo, throw.

class com.extensions.approval-test.approval.entity.ApprovalVirtualEntityCust implements
IApprovalVirtualEntity:

    {com/qad/approvals/dsApproval.i}
    {com/qad/approvals/entityv3/dsApprovalField.i}
    {com/qad/approvals/entityv3/dsApprovalFieldList.i}
    {com/qad/approvals/entityv3/dsEntityFieldValidation.i}
    {com/qad/approvals/entityv3/dsApprovalAction.i}
    {com/qad/approvals/config/dsApprovalConfiguration.i}
```

```
    {com/qad/qra/base/dsKeyField.i }
    {com/qad/qra/metadata/dsEntityMapping.i}

    define private variable logger as class ILogger no-undo.
    define variable virtualBe   as IVirtualBusinessEntity no-undo.
    define variable virtualBeDs as IInterfaceDatasets     no-undo.
    define variable config      as class IGenericApprovalConfiguration no-undo.

    constructor public ApprovalVirtualEntityCust ():
        assign virtualBe   = QraServices:GetVirtualBusinessEntity().
    end constructor.

    /*
    define public property GenericApprovalService as class IGenericApproval no-undo
        get:
            if not valid-object(GenericApprovalService) then do:
                if QraConfig:IsServiceAvailable ("com.qad.approvals.IGenericApproval") then
                    this-object:GenericApprovalService = GenericApprovalServices:
GetGenericApproval().
            end.
            return GenericApprovalService.
        end get.
        set.
    define public property CurrencyService as class ICurrency no-undo
        get:
            if not valid-object(CurrencyService) then
                this-object:CurrencyService = BaseServices:GetCurrency().
            return CurrencyService.
        end get.
        set.

    define public property ExchangeRateService as class IExchangeRate no-undo
        get:
            if not valid-object(ExchangeRateService) then
                this-object:ExchangeRateService = BaseServices:GetExchangeRate().
            return ExchangeRateService.
        end get.
        set.
    */


    method public void GetApprovalFields ( input dataset dsApprovalAction,
                                           output dataset dsApprovalField):
        define variable entityInstanceUri     as character   no-undo.
        define variable approverUserid        as character   no-undo.
        define variable dsEntity              as handle      no-undo.
        define variable bufferField           as handle no-undo.

message '## CUST:GetApprovalFields() -START' skip.
dataset dsApprovalAction:write-xml("FILE",
"Cust_GetApprovalFields_dsApprovalAction_VirtualApprovalEntity" + string(time) + ".xml", yes).
        dataset dsApprovalField:empty-dataset().

        for each ttApprovalAction on error undo, throw:
            entityInstanceUri = ttApprovalAction.EntityInstanceUri.
            approverUserid    = ttApprovalAction.ApproverActionUserid.

            this-object:VirtualBEFetch(input entityInstanceUri,
                                       output dataset-handle dsEntity by-reference).

            dsEntity:get-top-buffer():find-first() no-error.
            /*if dsEntity:get-top-buffer():available
            then do:
                bufferField = dsEntity:get-top-buffer():buffer-field(ttApprovalField.FieldName)
no-error.
                if bufferField:available then
                   bufferField:buffer-value = ttApprovalField.FieldValue.
            end.*/

            this-object:GetApprovalFieldList(input entityInstanceUri, output dataset
dsApprovalFieldList by-reference).

            for each ttApprovalFieldList
            on error undo, throw:
                bufferField = dsEntity:get-top-buffer():buffer-field(ttApprovalFieldList.
FieldName) no-error.
                if bufferField:available then do:
                    create ttApprovalField.
                    assign
```

```
                        ttApprovalField.EntityInstanceUri = entityInstanceUri
                        ttApprovalField.ApproverUserid    = approverUserid
                        ttApprovalField.FieldName         = ttApprovalFieldList.FieldName
                        ttApprovalField.FieldValue        = string(bufferField:buffer-value)
                        /*ttApprovalField.IsEnable          = pIsEnable
                        ttApprovalField.IsReadOnly        = pIsReadOnly
                        ttApprovalField.IsVisible         = pIsVisible
                        ttApprovalField.IsRequired        = pIsRequired
                        ttApprovalField.FieldLabel        = pFieldLabel
                        ttApprovalField.DataListCode      = pDataList
                        ttApprovalField.HelpId            = pHelp*/
                        .
                end.
            end.
        end.
dataset dsApprovalAction:write-xml("FILE",
"Cust_GetApprovalFields_dsApprovalField_VirtualApprovalEntity" + string(time) + ".xml", yes).

    end method.

    method public character GetApprovalHash (input entityInstanceUri as character):
        define variable dsEntity as handle no-undo.
        define variable approvalApi as class IGenericApproval no-undo.
        define variable fieldsList as character no-undo.

message '## CUST: GetApprovalHash() -START' skip.
        this-object:VirtualBEFetch(input entityInstanceUri ,
                                   output dataset-handle dsEntity by-reference).

        this-object:GetApprovalFieldList(input entityInstanceUri, output dataset
dsApprovalFieldList by-reference).

        fieldsList = "".
        for each ttApprovalFieldList on error undo, throw:
            fieldsList = if fieldsList = "" then ttApprovalFieldList.FieldName
                         else (fieldsList + "," + ttApprovalFieldList.FieldName).
        end.

        approvalApi = GenericApprovalServices:GetGenericApproval().

        return approvalApi:CalculateHash (input dataset-handle dsEntity, "", fieldsList).

    end method.

    method public character GetStakeholders (input entityInstanceUri as character):
message '## CUST: GetStakeholders() -START' skip.
    end method.

    method public void SetApprovalFields (input dataset dsApprovalField, input dataset
dsApprovalAction):
        define variable dsEntity     as handle no-undo.
        define variable transManager as ITransactionManager no-undo.
        define variable Xid          as character no-undo.
        define variable bufferField  as handle no-undo.
        define variable qry          as handle no-undo.
        define variable qryBuf       as handle no-undo.

message '## CUST: SetApprovalFields() -START' skip.
dataset dsApprovalField:write-xml("FILE",
"Cust_SetApprovalFields_dsApprovalField_VirtualApprovalEntity" + string(time) + ".xml", yes).
dataset dsApprovalAction:write-xml("FILE",
"Cust_SetApprovalFields_dsApprovalAction_VirtualApprovalEntity" + string(time) + ".xml", yes).

        transManager = QraServices:GetTransactionManager().

        for each ttApprovalField break by ttApprovalField.EntityInstanceUri on error undo, throw:

            if first(ttApprovalField.EntityInstanceUri) then
                Xid = transManager:StartTransaction().

            if first-of(ttApprovalField.EntityInstanceUri) then do:
                this-object:VirtualBEFetch(input ttApprovalField.EntityInstanceUri,
                                           output dataset-handle dsEntity by-reference).
dsEntity:write-xml("FILE", "Cust_SetApprovalFields_dsEntity" + string(time) + ".xml", yes).
            end.
```

```
/*
            /*SetApprovalFields*/
            dsEntity:get-top-buffer():find-first() no-error.
            if dsEntity:get-top-buffer():available
            then do:
                bufferField = dsEntity:get-top-buffer():buffer-field(ttApprovalField.FieldName)
no-error.
                if bufferField:available then do:
                    case bufferField:data-type:
                        when "decimal" then bufferField:buffer-value = decimal(ttApprovalField.
FieldValue).
                        when "date" then bufferField:buffer-value = date(ttApprovalField.
FieldValue).
                        when "integer" then bufferField:buffer-value = integer(ttApprovalField.
FieldValue).
                        when "logical" then bufferField:buffer-value = logical(ttApprovalField.
FieldValue).
                    end case.
                end.
                    /*bufferField:buffer-value = if bufferField:data-type = "decimal" then decimal
(ttApprovalField.FieldValue)
                                            else if bufferField:data-type = "integer" then
integer(ttApprovalField.FieldValue)
                                            else if bufferField:data-type = "datetime" then date
(ttApprovalField.FieldValue)
                                            else ttApprovalField.FieldValue.*/
            end.
            if last-of(ttApprovalField.EntityInstanceUri) then do:
                this-object:VirtualBEUpdate(input ttApprovalField.EntityInstanceUri,
                                        input dataset-handle dsEntity by-reference).
            end.
            finally:
                if valid-handle(qry) and
                   qry:is-open
                then qry:query-close().

                delete object qry no-error.
            end finally.
*/
        end.
        if Xid <> "" then do on error undo, throw:
            transManager:Commit(Xid).

            catch e as Progress.Lang.Error:
                transManager:Rollback(Xid).
                undo, throw e.
            end catch.
        end.

    end method.

    method public void SetApprovalResult (input dataset dsApproval):
        define variable dsEntity     as handle no-undo.
        define variable transManager as ITransactionManager no-undo.
        define variable Xid          as character no-undo.
        define variable bufferField  as handle no-undo.

message '## CUST: SetApprovalResult() -START' skip.
dataset dsApproval:write-xml("FILE", "Cust_SetApprovalResult_dsApproval_VirtualApprovalEntity" +
string(time) + ".xml", yes).

        transManager = QraServices:GetTransactionManager().
        Xid = transManager:StartTransaction().

        do on error undo, throw:
            for each ttApproval no-lock on error undo, throw:

                do on error undo, leave:

                    this-object:VirtualBEFetch(input ttApproval.EntityInstanceUri,
                                            output dataset-handle dsEntity by-reference).

dsEntity:write-xml("FILE", "Cust_SetApprovalResult_dsEntity_Fetch" + string(time) + ".xml", yes).
                    catch e as Progress.Lang.Error:
                       logger:Error(e).
                       return.
                    end catch.

                end.
```

```
                dsEntity:get-top-buffer():find-first() no-error.
                if dsEntity:get-top-buffer():available
                then do:
                    bufferField = dsEntity:get-top-buffer():buffer-field("ExpenseStatus") no-
error.
                    if bufferField:available then
                       bufferField:buffer-value = ttApproval.ApprovalStatus.
                       /* bufferField:buffer-value = GenericApprovalConstants:STATUS_APPROVED. */
                end.

dsEntity:write-xml("FILE", "Cust_SetApprovalResult_dsEntity_Update" + string(time) + ".xml", yes).
                this-object:VirtualBEUpdate(input ttApproval.EntityInstanceUri,
                                            input dataset-handle dsEntity by-reference).

            end.

            transManager:Commit(Xid).

            catch e as Progress.Lang.Error :
                transManager:Rollback(Xid).
                undo, throw e.
            end catch.
        end.

    end method.

    method public void ValidateEntityFields (input dataset dsApprovalField, output dataset
dsEntityFieldValidation):
        define variable dsEntity            as handle no-undo.
        define variable entityInstanceUri   as character   no-undo.
        define variable bufferField         as handle no-undo.
        define variable totAmt              as decimal no-undo.

message '## CUST:ValidateEntityFields() -START' skip.
dataset dsApprovalField:write-xml("FILE",
"Cust_ValidateEntityFields_dsApprovalField_VirtualApprovalEntity" + string(time) + ".xml", yes).
        dataset dsEntityFieldValidation:empty-dataset().

        for each ttApprovalField break by ttApprovalField.EntityInstanceUri on error undo, throw:
            if first-of(ttApprovalField.EntityInstanceUri) then do:
                entityInstanceUri = ttApprovalField.EntityInstanceUri.
                this-object:VirtualBEFetch(input ttApprovalField.EntityInstanceUri,
                                           output dataset-handle dsEntity by-reference).
            end.

            dsEntity:get-top-buffer():find-first() no-error.
            if dsEntity:get-top-buffer():available
            then
                bufferField = dsEntity:get-top-buffer():buffer-field(ttApprovalField.FieldName)
no-error.

            /* Validations on the Additional Approval fields */
            case ttApprovalField.FieldName:
                when "TotalAmount" then do:
                    totAmt = decimal(ttApprovalField.FieldValue).
                    if totAmt > 1000 then do:
                        create ttEntityFieldValidation.
                        assign
                            ttEntityFieldValidation.EntityInstanceUri = entityInstanceUri
                            ttEntityFieldValidation.FieldName = ttApprovalField.FieldName
                            ttEntityFieldValidation.ValidationMessage = "TotalAmount is > 1000".
                    end.
                    if totAmt > 5000 then do:
                        create ttEntityFieldValidation.
                        assign
                            ttEntityFieldValidation.EntityInstanceUri = entityInstanceUri
                            ttEntityFieldValidation.FieldName = ttApprovalField.FieldName
                            ttEntityFieldValidation.ValidationMessage = "TotalAmount is > 5000".
                    end.
                end.
            end case.
        end.

    end method.

    method public void GetCurrencyConvertedEntity (input entityInstanceUri as character, input
currencyCode as character, output dataset-handle dsEntity):
        define variable bufferCurr  as handle no-undo.
```

```
        define variable bufferAmt  as handle no-undo.
message '## CUST:GetCurrencyConvertedEntity() -START' skip.
        this-object:VirtualBEFetch(input entityInstanceUri,
                                   output dataset-handle dsEntity by-reference).
                dsEntity:get-top-buffer():find-first() no-error.
                if dsEntity:get-top-buffer():available
                then do:
                    assign
                        bufferCurr = dsEntity:get-top-buffer():buffer-field("CurrencyCode").
                        bufferAmt = dsEntity:get-top-buffer():buffer-field("Amount")
                        no-error.
                    if bufferAmt:available and
                       bufferCurr:available and
                       bufferCurr:buffer-value <> currencyCode
                    then do:
                       /*convert amount*/
                       /* bufferAmt:buffer-value = bufferAmt:buffer-value * 2. */
                    end.
                end.

    end method.

    method public logical isValidCurrency(input currencyCode as character):
        define variable listAllowCurr as character no-undo.
message '## CUST:isValidCurrency() -START' skip.
        listAllowCurr = "USD,EUR".
        /* return (index(listAllowCurr, currencyCode) > 0). */
        return true.
    end method.

    /*
    method public logical isValidCurrency (input currencyCode as character ):
        define variable isValidCurency as logical    no-undo.
        isValidCurency = this-object:CurrencyService:IsValidActiveCurrency (input currencyCode).
        return isValidCurency.
    end method.
    */

    method public void GetApprovalFieldList (input entityInstanceUri as character, output dataset
dsApprovalFieldList):

        define variable entityUri             as character   no-undo.

        entityUri         = this-object:getEntityUri(entityInstanceUri).

        dataset dsApprovalFieldList:empty-dataset().
        dataset dsApprovalConfiguration:empty-dataset().

        config = GenericApprovalServices:GetGenericApprovalConfiguration().
        config:Fetch(input entityUri, output dataset dsApprovalConfiguration by-reference).

        for each ttApprovalConfigurationField on error undo, throw:
            create ttApprovalFieldList.
            assign
                ttApprovalFieldList.EntityInstanceUri = entityInstanceUri
                ttApprovalFieldList.FieldName         = ttApprovalConfigurationField.FieldName.
                .
        end.
    end method.

    method public void VirtualBEFetch(input entityInstanceUri as character,
                                      output dataset-handle dsEntityVirtual):

        define variable entityURI as character no-undo.

        this-object:ParseInstanceUri(input entityInstanceUri,
                                      output EntityURI,
                                      output dataset dsKeyField by-reference).
        virtualBe:Fetch(
            input entityURI,
            input dataset dsKeyField,
            output dataset-handle dsEntityVirtual by-reference).
    end.

    method public logical VirtualBEExists(input entityInstanceUri as character):

        define variable entityURI as character no-undo.

        this-object:ParseInstanceUri(input entityInstanceUri,
```

```
                                    output EntityURI,
                                    output dataset dsKeyField by-reference).
        return virtualBe:Exists(
            input entityURI,
            input dataset dsKeyField).
    end.

    method public void VirtualBEUpdate(input entityInstanceUri as character,
                                     input dataset-handle dsEntityVirtual):
        define variable entityURI as character no-undo.

        this-object:ParseInstanceUri(input entityInstanceUri,
                                    output EntityURI,
                                    output dataset dsKeyField by-reference).
        virtualBe:Update(
            input entityURI,
            input dataset-handle dsEntityVirtual by-reference).
    end.

    method public void VirtualBECreate(input entityInstanceUri as character,
                                     input dataset-handle dsEntityVirtual):
        define variable entityURI as character no-undo.

        this-object:ParseInstanceUri(input entityInstanceUri,
                                    output EntityURI,
                                    output dataset dsKeyField by-reference).
        virtualBe:Create(
            input entityURI,
            input dataset-handle dsEntityVirtual by-reference).
    end.

    method public void VirtualBEDelete(input entityInstanceUri as character,
                                     input dataset-handle dsEntityVirtual):
        define variable entityURI as character no-undo.

        this-object:ParseInstanceUri(input entityInstanceUri,
                                    output EntityURI,
                                    output dataset dsKeyField by-reference).
        virtualBe:Delete(
            input entityURI,
            input dataset-handle dsEntityVirtual by-reference).
    end.

    /**
     * Parse a URI for an Virtual business entity.
     * This will take the URI and KeyFields value from entityInstanceUri"
     */
    method public void ParseInstanceUri (input entityInstanceUri as character,
                                         output entityURI as character,
                                         output dataset dsKeyField):

        define variable uriResolver     as class IResourceResolver no-undo.
        define variable entityMapping   as class IEntityMapping no-undo.
        define variable businessKeys    as class IList no-undo.
        define variable mappingKeyFields as character no-undo.
        define variable iA              as integer no-undo.

        uriResolver     = QraServices:GetResourceResolver().
        entityMapping   = QraServices:GetEntityMappingService().

        uriResolver:SetUri (entityInstanceUri).
        assign entityURI = "urn:" + uriResolver:GetType() mappingKeyFields = "".
message "## entityInstanceUri = "  entityInstanceUri skip.
message "## entityUri=" entityURI skip.
         /* Check if it's a "be:" resource */
        if index(entityURI, "be:") = 0 then
            undo, throw new QraError(new Message(QraMessageKey:INVALID_URI, entityURI)).

        /*GetEntityMapping*/
        dataset dsEntityMapping:empty-dataset().
        entityMapping:Fetch(input entityURI,
            output dataset dsEntityMapping by-reference).

        find first ttEntityMapping
            where ttEntityMapping.EntityUri = entityURI
            no-error.

        if available ttEntityMapping then
            assign mappingKeyFields = ttEntityMapping.KeyFields.
```

```
        else
            undo, throw new QraError(new Message(QraMessageKey:ENTITY_MAPPING_DOES_NOT_EXIST,
entityURI)).

        businessKeys = uriResolver:ParseIdentifier().
        if businessKeys:Count <> num-entries(mappingKeyFields) then
            undo, throw new QraError (new Message(QraMessageKey:INVALID_URI, entityURI)).

        dataset dsKeyField:empty-dataset().
        do iA = 1 to businessKeys:Count:
            create ttKeyField.
            assign ttKeyField.KeyFieldName  = entry(iA, mappingKeyFields)
                   ttKeyField.KeyFieldValue = businessKeys:Get(iA):ToString().
        end.

    end method.

    method public character getEntityUri (input entityInstanceUri as character):

        define variable uriResolver     as class IResourceResolver no-undo.

        uriResolver     = QraServices:GetResourceResolver().
        uriResolver:SetUri (entityInstanceUri).
        return uriResolver:GetType().

    end method.

    method public character CreateApprovalField (
        input pEntityUri as character,
        input pUserId as character,
        input pFieldName as character,
        input pFieldValue as character,
        input pIsEnable as logical,
        input pIsReadOnly as logical,
        input pIsVisible as logical,
        input pIsRequired as logical,
        input pFieldLabel as character,
        input pDataList as character,
        input pHelp as character,
        input-output dataset dsApprovalField):

        create ttApprovalField.
        assign
            ttApprovalField.EntityInstanceUri = pEntityUri
            ttApprovalField.ApproverUserid    = pUserId
            ttApprovalField.FieldName         = pFieldName
            ttApprovalField.FieldValue        = pFieldValue
            ttApprovalField.IsEnable          = pIsEnable
            ttApprovalField.IsReadOnly        = pIsReadOnly
            ttApprovalField.IsVisible         = pIsVisible
            ttApprovalField.IsRequired        = pIsRequired
            ttApprovalField.FieldLabel        = pFieldLabel
            ttApprovalField.DataListCode      = pDataList
            ttApprovalField.HelpId            = pHelp
            .

    end method.
end class.
```

D. Compile the code

In your environment, run the following command:

yab dev-<extractfolder>-update

In our case, that is:

yab dev-approval-update

```
                dev-approval-update (14 tasks)                    [APPLY]
-------------------------------------------------------------------------------
1/14  database-dev-approval-extension-structure-list        OK (0.004 s)
2/14  database-dev-approval-extension-structure-file-update OK (0.013 s)
3/14  database-dev-approval-extension-structure-validate    OK (0.002 s)
4/14  database-dev-approval-extension-create                ADDED (23.709 s)
5/14  database-dev-approval-extension-structure-update      OK (0.003 s)
6/14  database-dev-approval-extension-schema-update         UPDATED (12.693 s)
```

```
7/14  database-dev-approval-extension-schema-snapshot       OK (0.069 s)
8/14  database-dev-approval-extension-data-xml-update       OK (0.007 s)
9/14  database-dev-approval-extension-data-dotd-update      OK (0.082 s)
10/14 dev-approval-init-check                               OK (0.001 s)
11/14 code-dev-approval-db-start-stop                       SKIPPED (0.001 s)
12/14 code-dev-approval-update                              OK (0.502 s)
13/14 code-dev-approval-db-start-stop                       SKIPPED (0.000 s)
14/14 prolib-dev-approval-create                            OK (0.014 s)
-------------------------------------------------------------------------------

BUILD SUCCESSFUL (45.037 s)
```

E. Add this new implementation of IApprovalVirtualEntityCust to module-config.xml

Go to the folder approval/config and add a new IVirtualBusinessEntity to module-config.xml.

module-config.xml

```
<Module>
    <ModuleInfo
                Key="extensions.approval-test"
                Version="@module.version@"
                Vendor="@module.vendor@"
                VendorUrl="@module.vendorurl@"
                DisplayName=""
                Description=""
                Date="@module.date@"
                ClassName="com.extensions.approval-test.Module"
                Uri="urn:app:com.extensions.approval-test"/>

    <Service
                ServiceClass="com.qad.approvals.entityv3.IApprovalVirtualEntity"
                ServiceKey="virtualbe"
                ImplClass="com.extensions.approval-test.approval.entity.
ApprovalVirtualEntityCust"
                LifetimePolicy="factory"/>

    <Propath Path="/dr01/qadapps/systest/approval/bin/approval.pl"/>
</Module>
```

F. Register the new code in your environment

In your environment, run the following command:

yab dev-<extractfolder>-code-register

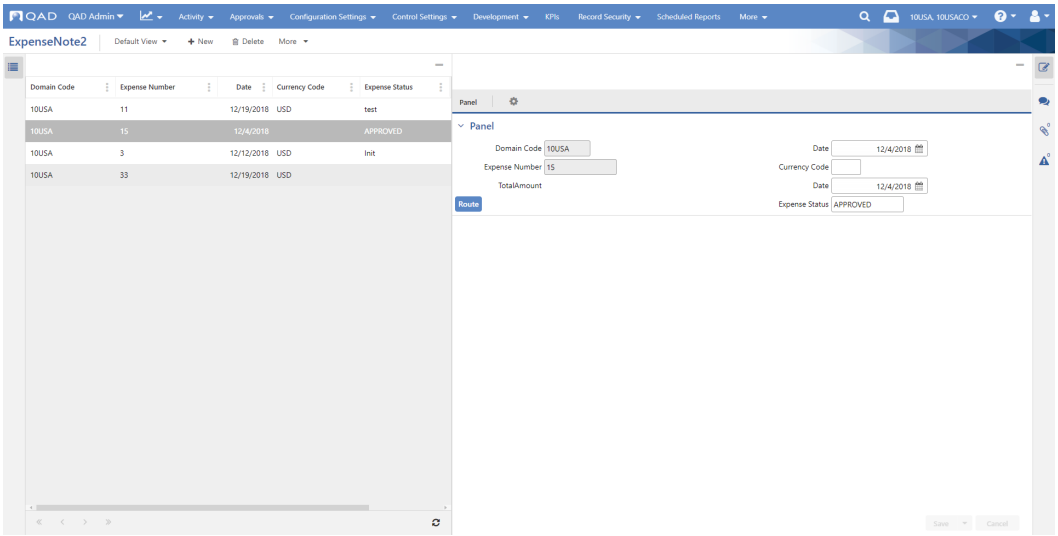
In our case, that is:

yab dev-approval-code-register

```
             dev-approval-code-register (10 tasks)                  [APPLY]
-------------------------------------------------------------------------------
1/10  dev-approval-code-register                            OK (0.076 s)
2/10  appserver-crm-trim                                    OK (0.714 s)
3/10  appserver-eam-trim                                    OK (0.609 s)
4/10  appserver-fin-trim                                    OK (0.307 s)
5/10  appserver-grs-trim                                    OK (0.307 s)
6/10  appserver-mfg-trim                                    OK (0.716 s)
7/10  appserver-qra-trim                                    OK (0.610 s)
8/10  appserver-qxosi-trim                                  OK (0.306 s)
9/10  appserver-qxoui-trim                                  OK (0.305 s)
10/10 appserver-qxtnative-trim                              OK (0.306 s)
-------------------------------------------------------------------------------

BUILD SUCCESSFUL (4.908 s)
```

G. Test customization

Next, approved records will have the APPROVED status in the **Expense Status** field.


QAD
QAD Admin
Activity
Approvals
Configuration Settings
Control Settings
Development
KPIs
Record Security
Scheduled Reports
More
10USA, 10USACO
ExpenseNote2
Default View
New
Delete
More
Domain Code
Expense Number
Date
Currency Code
Expense Status
10USA 11 12/19/2018 USD test
10USA 15 12/4/2018 APPROVED
10USA 3 12/12/2018 USD Init
10USA 33 12/19/2018 USD
Panel
Panel
Domain Code 10USA
Expense Number 15
TotalAmount
Date 12/4/2018
Currency Code
Date 12/4/2018
Expense Status APPROVED
Route
Save
Cancel

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.

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. Search for the business component that you have created.
2. On the Form panel 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

- Display Label
- Business Component
- Detail Table
- Physical Field
- Max Characters
- Format
- Default Value
- Required
- Element Name
- Control Type
- Input Width
- Input Height
- Label Visibility
- Label Width
- State
- Column Span
- Row Span

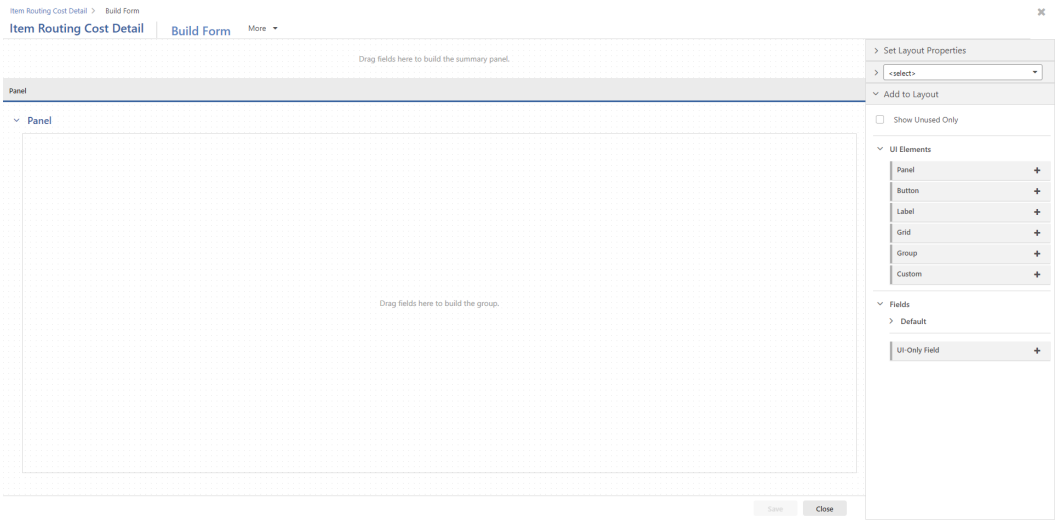
- Add to Layout
- Field Properties

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.

Linked View

Click Select Linked View to open the Business Component Views dialog and select the view to link the form to. The selected linked view will provide Drill Downs, Activity, and Attachments for the form.

Allow Drill Downs on Form

Indicates whether contextual drill downs are enabled for the form (Hybrids, Standalone Forms, and Individual Actions). When enabled, linked view drill downs appear on the Drill-Down Links panel on the right when you click in the form.

Allow Activity on Form

Indicates whether contextual activity is enabled for the form (Hybrids, Standalone Forms, and Individual Actions). When enabled, linked view activity appears on the Activity panel on the right when you click in the form. This option is activated only when you select the Activity Panel checkbox on the Main panel under Views.

Allow Attachments on Form

Indicates whether contextual attachments are enabled for the form (Hybrids, Standalone Forms, and Individual Actions). When enabled, linked view attachments appear on the Attachments panel on the right when you click in the form. This option is activated only when you select the Attachments Panel checkbox on the Main panel under Views.

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.

Input Height

Indicates the height available for the field's data input. You can set Input Height to as high as you want, there is no restriction. The minimum Input Height allowed is 1. Default height: Format (or max characters)/60, up to a max of 9 textarea rows and a row span of 5.

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.

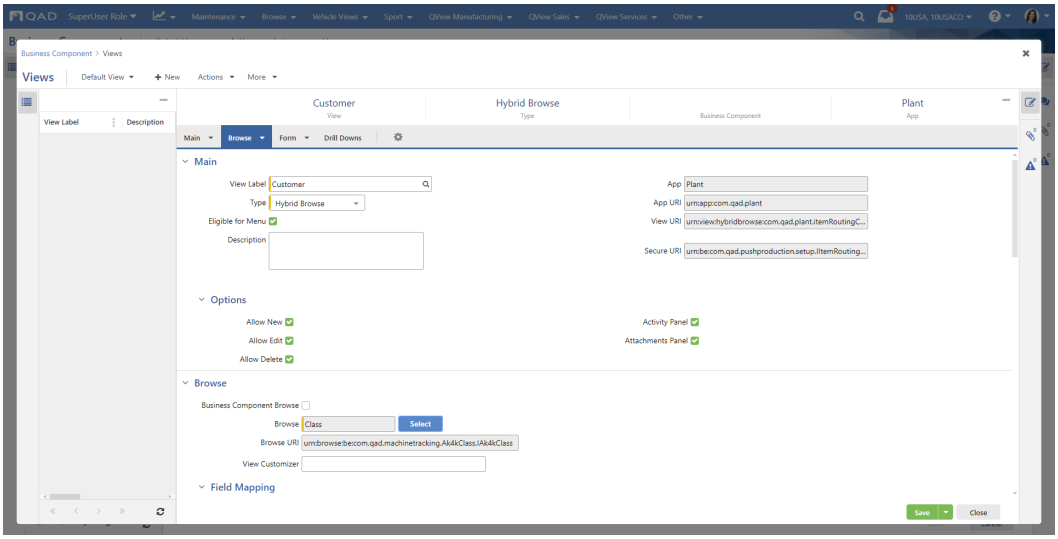
The URI of a business component browse has the pattern: urn:browse:be:*

The BC browse can include the following fields:

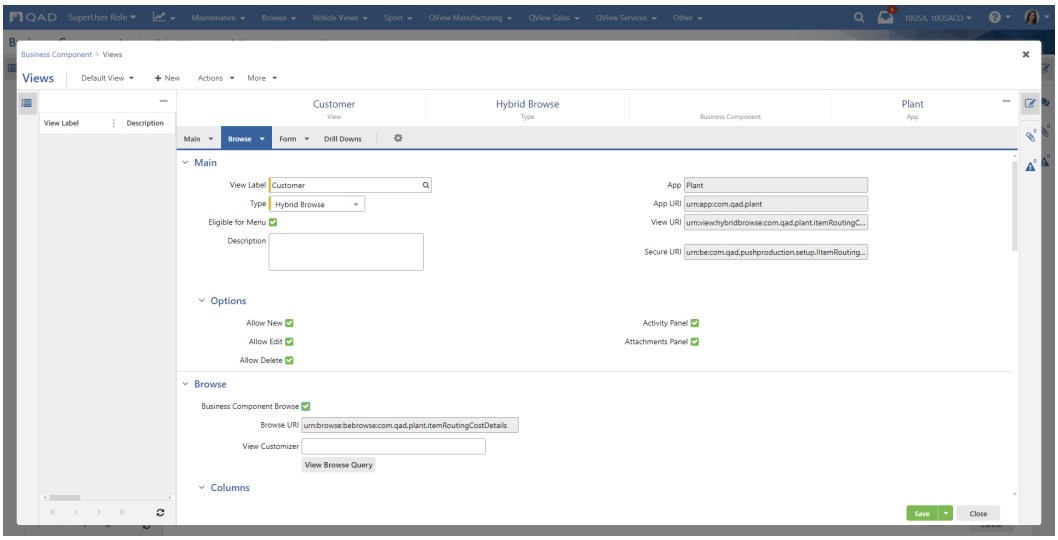
- Fields from the main table of the BC
- Fields of embedded BCs for which there is a child-parent relationship
- Fields of the main table of BCs that are related in One-to-one or Many-to-one fashion
- The key fields of the main table of the BC are always selected, and they cannot be removed from the 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 platform 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).

Description

A short description for the view. This field is optional.

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

Allow New

When selected, the New button is displayed in the runtime with drill downs and the + icon.

Allow Edit

When selected, the Save and Cancel buttons are displayed in the runtime and you can edit the records. If you clear the checkbox, the Save and Cancel buttons will be removed, and the Edit button changes to Open.

Allow Delete

When selected, the Delete button is displayed in the runtime, which allows users to delete records.

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 the Business Component Browse checkbox 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).

View Browse Query

Provides visibility into the SQL query used to execute the BC browse so that an administrator can better understand how the browse is defined and work with end users to explain the browse data displayed and clarify the end users' data expectations. This field is displayed only if the Business Component Browse checkbox is selected. When you click the View Browse Query button, the Browse Query pop-up window opens with the SQL statement. Only the fields that are selected in the browse's Columns grid are included in the query statement. JOIN appears if you create a BC browse with related fields.

Browse query is in read-only mode. It is provided only for the visualization of the SQL and is not intended for entering the SQL manually.

You can copy the SQL query and run it from other applications to see the result table.

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

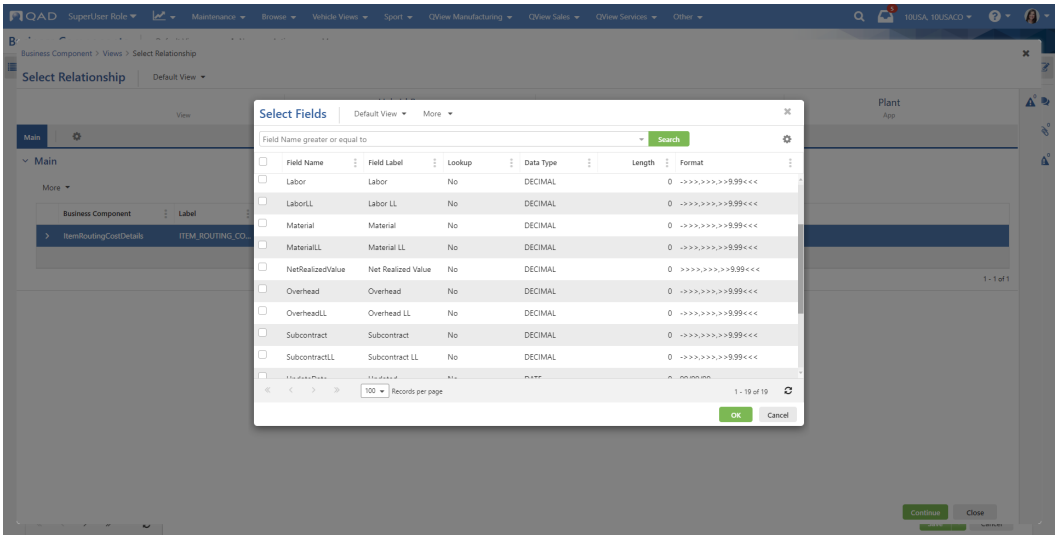
Specifies the corresponding key field on the browse.

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, which are added to the Form. You can remove columns by selecting the checkbox for each corresponding field. New additional fields can be added to the Columns grid by using the +Select button.

Also, you can add browse columns from related business components. When you click the +Select button, the Select Relationship pop-up window opens. When you expend the relationship, it will show the relationships up to four levels. You can select the relationship you need, click Continue, and then in the Select Fields dialog, add columns/fields from this relationship.

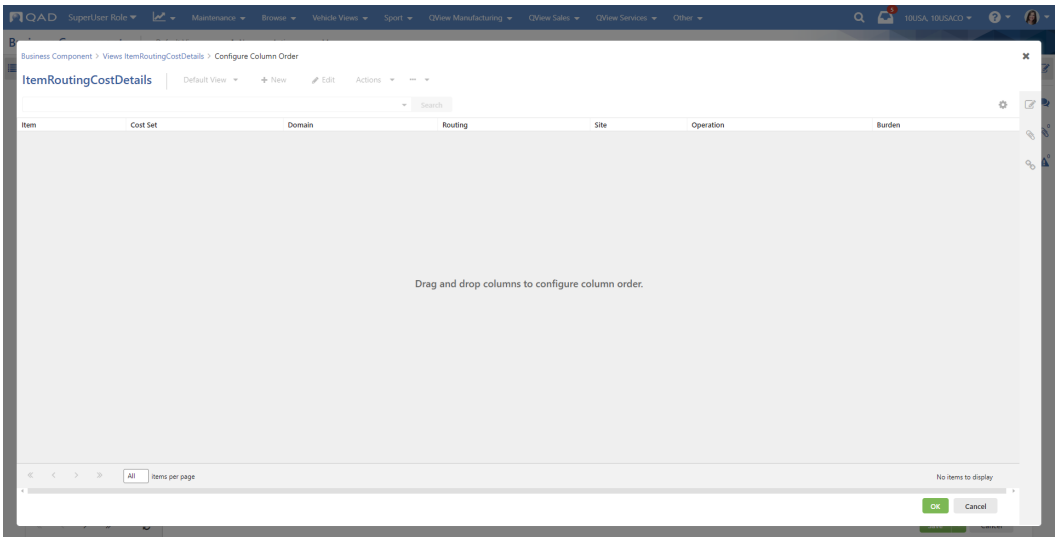


Column Order panel

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.



Predefined Search Criteria panel

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

It provides the ability to define complex filter logic for views by using AND and OR operators, as well as hide browse filters so that they are not exposed to the users.

When creating or editing the view, you must first save the view before you can update the predefined search criteria. The predefined search criteria are disabled until you save the view. Note that you cannot remove the browse columns that are used in the predefined search criteria.

For example, you can add the following predefined search criteria: "Opportunity.Owner equals Current User". Click **Include Field** to select fields, click **Include Operator** to select an operator to apply to the field, and click **Include Variable** to select a date variable, etc. Note that Fiscal Date Variables are available if Fiscal Calendar is installed in the system. Click **Check Syntax** to validate the syntax of your search criteria. The added search filter is visible in the advanced search under the Predefined Criteria menu if enabled.

Show in Advanced Search

Specifies whether the predefined search criteria are visible to the users in the advanced search.

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 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. The following drill down types are available:

- **Business Component Relationship** are the drill downs that are generated automatically when Business Component relationships are defined, including lookups. This applies to both coded and platform BCs.
- **.NET UI - Browse Based** are the legacy .Net drill downs that exist in the .Net today and will only be present for coded BCs.
- **User Added - Business Component Based** are the drill downs that are added using the entity metadata for the drill down definition, mapping, and conditions. These are the drill downs that are manually added starting with the March 2020 release.
- **User Added - Browse Based** are the drill downs that are added using the browse metadata for the drill down definition, mapping, and conditions. These are the drill downs that are manually added prior to the March 2020 release.

You can view the drill down type in the Views > Drill Downs panel > Generated From column and in the Drill Downs detail pop-up window.

The following drill down types are visible from linked grids and actions:

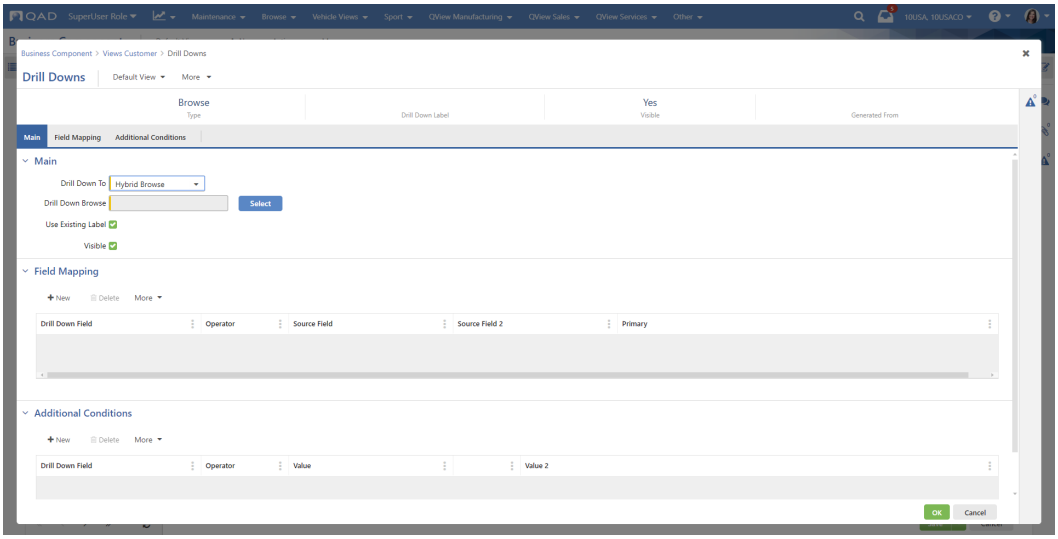
- Business Component Relationship
- User Added - Business Component Based

The following drill down types are not visible from linked grids and actions:

- .NET UI - Browse Based
- User Added - Browse Based

In the Views > Drill Downs panel > Warnings column, you can see a warning icon next to the drill downs, which have visibility issues.

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, you will see an empty modal window in the runtime.

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 drill down browse fields with source business component 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.

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.

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.

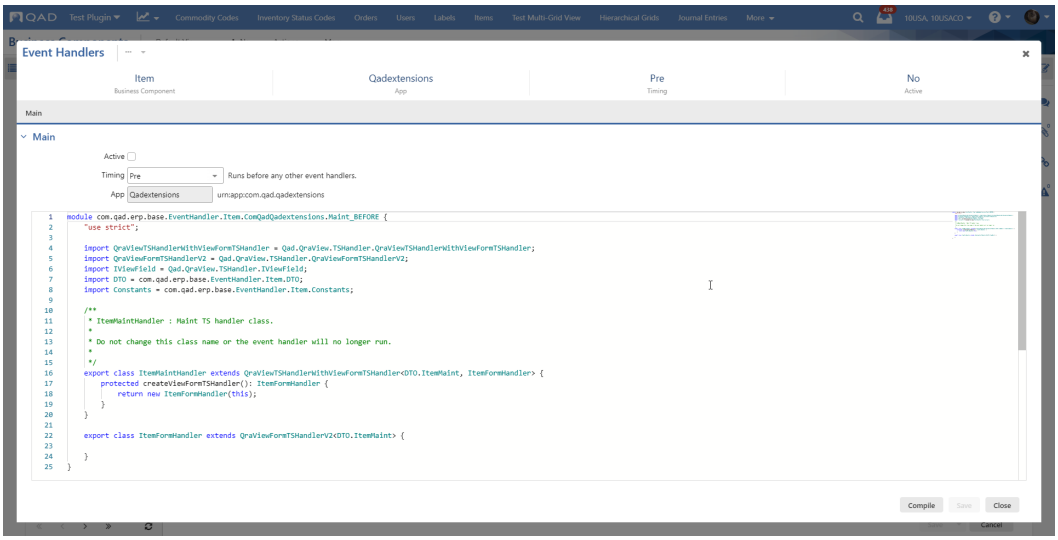
What is an Event Handler?

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

There are three types of Event Handlers:

1. Primary — primary event handler for the corresponding business component. It can be created only for platform 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 platform BC	Definition is in same App	Behavior
Add “pre/post” event handler	yes	yes	Not possible
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 a platform one 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 a platform one 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. Fields from embedded business components are also available in an external grid when that grid represents records for a business component extended with a One-to-one relationship.

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 Form Builder, when you add a grid, the Form 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 child-parent 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.

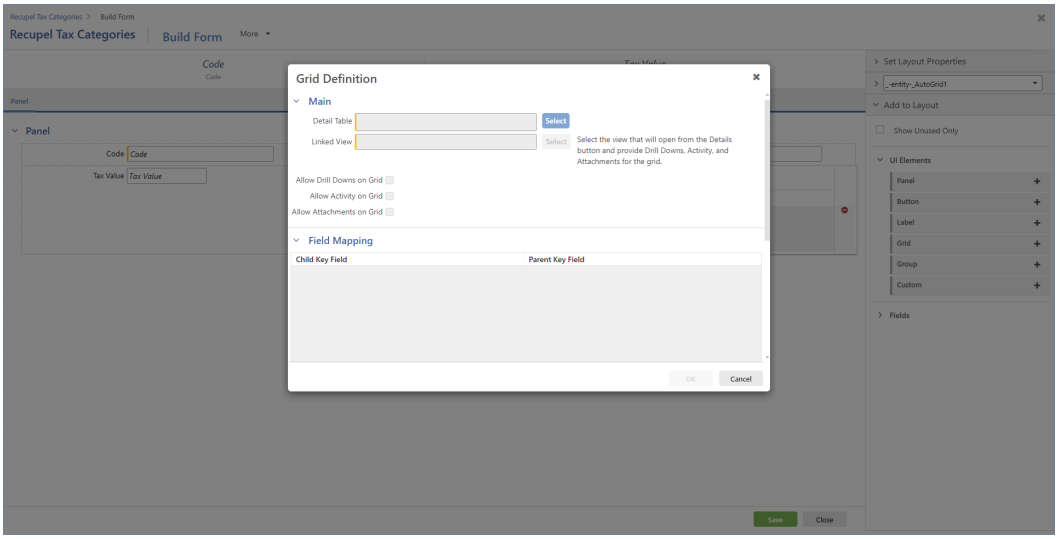
Currently, only external grids can be recursive hierarchical grids. Internal recursive hierarchical grids are not supported at this time.

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:



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.

Linked View

For an internal grid, select the view that will provide Drill Downs, Activity, and Attachments for the grid.

For an external grid, select the view that will open from the Details button and provide Drill Downs, Activity, and Attachments for the grid.

Allow Drill Downs on Grid

Indicates whether contextual drill downs are enabled for the grid. When enabled, linked view drill downs appear on the Drill-Down Links panel on the right when you click in the grid.

Allow Activity on Grid

Indicates whether contextual activity is enabled for the grid. When enabled, linked view activity appears on the Activity panel on the right when you click in the grid. This option is activated only when you select the Activity Panel checkbox on the Main panel under Views.

Allow Attachments on Grid

Indicates whether contextual attachments are enabled for the grid. When enabled, linked view attachments appear on the Attachments panel on the right when you click in the grid. This option is activated only when you select the Attachments Panel checkbox on the Main panel under Views.

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

Brings up a lookup with all related business component fields. This allows you to select fields from up to two levels of relationships.

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.

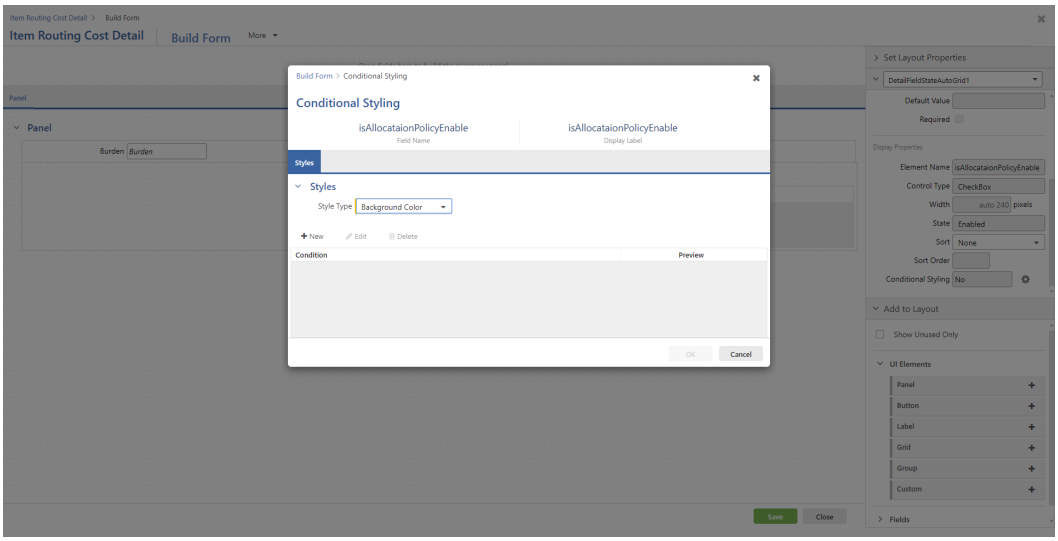
Form Builder - Build Form - Grid Conditional Styling

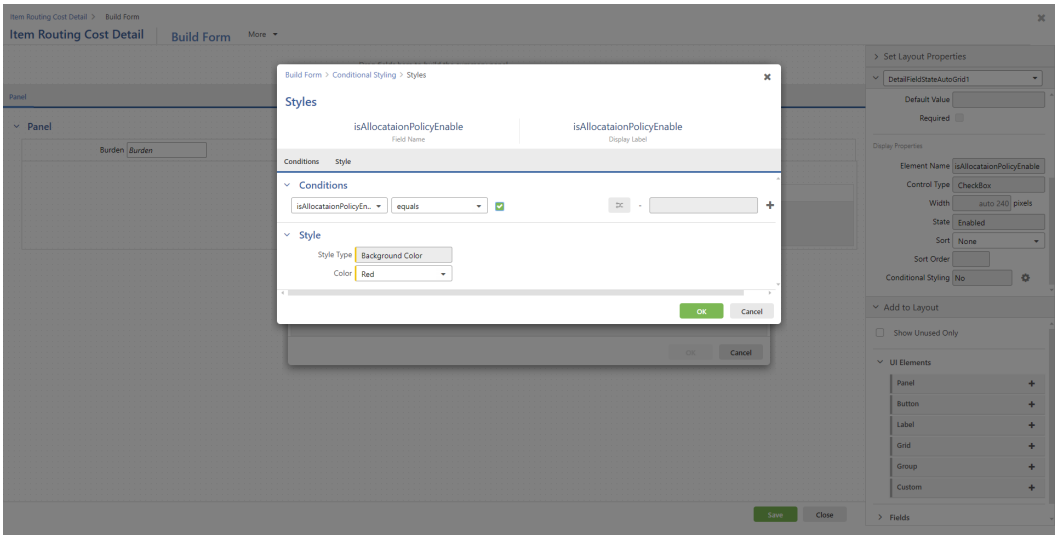
Introduction

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.

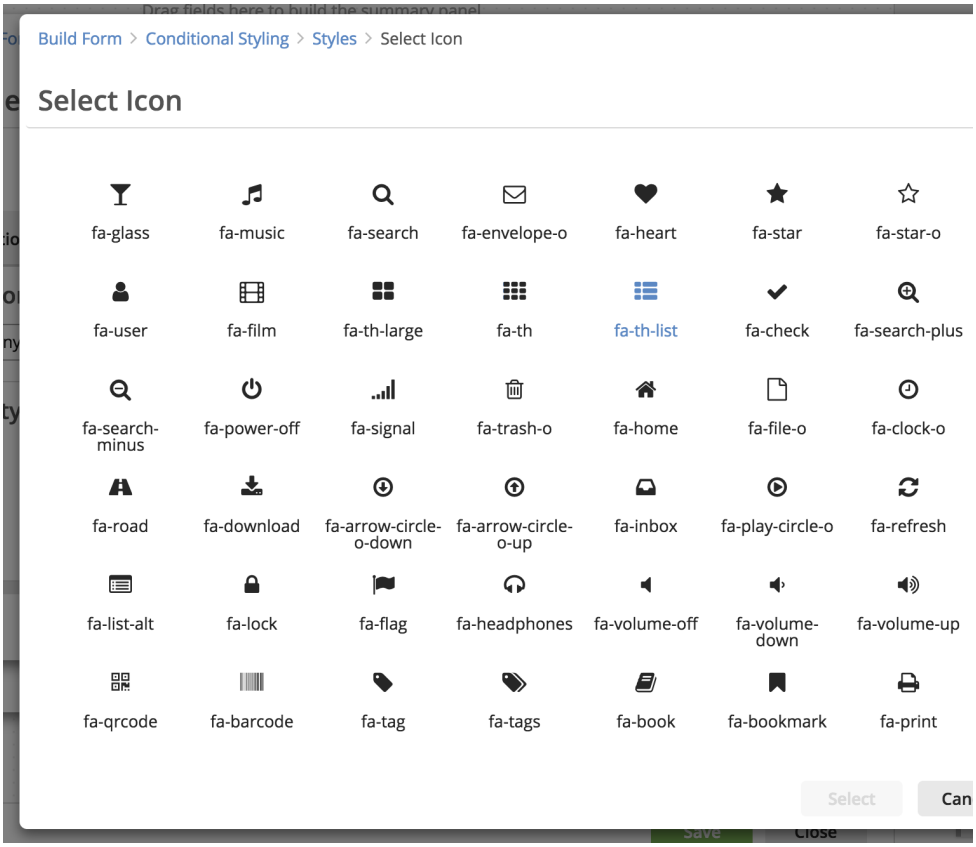
Style

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.

Form Builder Changes That Affect Design Layout

Introduction

When customers make changes in the Design Layout in one release, and then QAD makes changes to the business component's form with the next release, it may affect customers' Design Layouts with an upgrade. In this case, affected elements in the Design Layout will be relocated to the end of the area where they were previously located in the Design Layout.

This article includes guidelines on what changes QAD development teams should be aware of in the Form Builder that may potentially impact customers' Design Layouts during an upgrade, and what the impact is to customers if QAD development teams make those changes.

Form Builder Changes That Affect Design Layout

The following changes made in the Form Builder may impact customers' Design Layouts:

- Move, delete, or rename a field.
- Move, delete, or rename a panel.
- Move, delete, or rename a subpanel.
- Move, delete, or rename a button.
- Move, delete, or rename a label.
- Move, delete, or rename a grid.
- Move, delete, or rename a group.
- Move, delete, or rename a custom element.

Note that only those elements are affected that were changed (or dependent) both in the Form Builder and Design Layout.

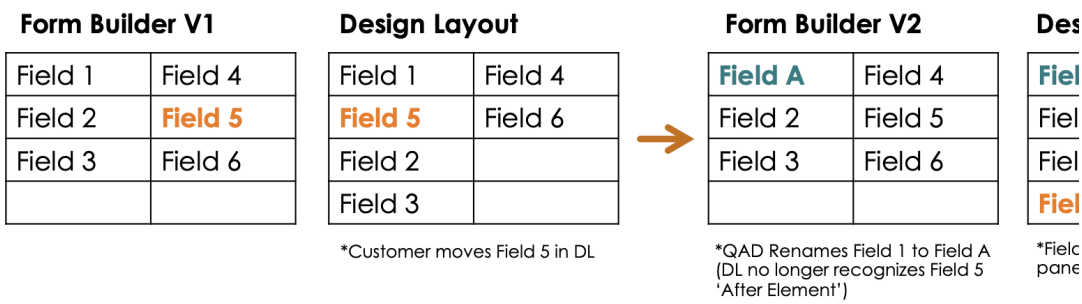
What is the Impact?

When customers make changes to the elements in the Design Layout, a new position of the elements is recorded in the code with the following properties:

- after — The element after which the changed element goes (called the After Element).
- column — The column of the changed element.
- element name — The name of the changed element.
- parent — The panel to which the changed element belongs.

After the changes in the Form Builder, the elements changed in the Design Layout may lose these properties, so they are relocated to the end of the area where they were previously located in the Design Layout. The elements will still remain in the correct column and panel.

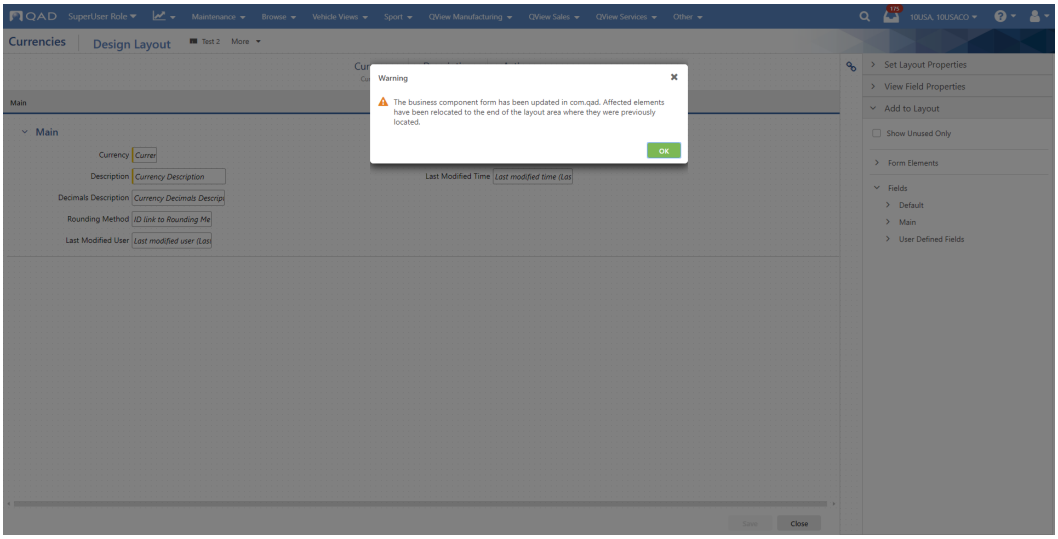
If the Design Layout cannot identify the After Element, the Design Layout changed element will be relocated to the end of the panel column it was placed in in the Design Layout. If the element is a panel that can no longer identify the correct placement in Design Layout, then it will be moved to the end of the form. Example:



You can find more examples here.

According to this approach, the layout elements may be slightly relocated. The relocation will be at the end of the area they were originally placed. So while the element (field/panel/subpanel/button/label/grid/group/custom element) order may be a little different, the elements will still be within a close proximity of their original placement. This will minimize the impact on customers and help with Design Layout re-adjustments.

When the Design Layout is affected, you can see the following warning message when you access the Design Layout:



FAQ

Q: Have I already made changes in the Form Builder that will impact a customer's Design Layout?

A: Compare your forms, either visually or using a code diff tool (diff the view metadata) to see if you have made any of the changes that could impact Design Layout. Keep in mind, even if you have made changes, it still will only impact a customers' layout if they have also made changes in Design Layout that are dependent on the elements you changed.

Q: How do I make sure my team considers this in the future when making changes in the Form Builder?

A: Developers can find the information in this article of the QAD Enterprise Platform Developer Guide.

Q: When is it acceptable to make a change that may impact Design Layout?

A: These types of changes should only be made when there are strong business reasons. If these changes are necessary, it is OK to make them. To mitigate impact to customers:

- Design forms with a larger vision in mind (as much as you can) so you can name and place elements in a more permanent position from the beginning.
- Group changes together into a release instead of making smaller changes across multiple releases.

Q: What are the plans to mitigate this issue in the future?

A: There are currently no plans to fix this, as it will require significant re-engineering in Design Layout (if it is even possible). To reduce the impact, we hope to implement the following:

- Enhance the current warning message to provide more visibility into which elements are affected and the type of Design Layout change that happened. This will help the IT Admins remake those changes, as it may have been a while since they last made updates there and would not remember exactly what they changed.
- Generate an Inbox notification to anyone who has QAD Admin access informing that some changes cannot be loaded in an active Design Layout the first time the screen is accessed (after upgrade) at runtime. The goal is to make QAD Admins aware of the issue early so they can investigate and address any issues early.

Examples - Form Builder Changes That Affect Design Layout

Introduction

This section provides a few additional examples on how the Form Builder changes affect the Design Layout. For more information about this topic, see Form Builder Changes That Affect Design Layout.

Moving Elements

Case 1

Form Builder v1		Design Layout v1		Form Builder v2		Design Layout v2	
Panel1		Panel1		Panel1		Panel1	
E1	E4	**E1**	E4	E2	E4	E2	E4
E2	**E5**	**E5**	E6	E3	**E5**	E3	E6
E3	E6	E2			E6	**E5**	
		E3					
Panel2		Panel2		Panel2		Panel2	
					E1		**E1**
		*Move E5 after E1		*Move E1		*E5 relocated	

Case 2

Form Builder v1		Design Layout v1		Form Builder v2		Design Layout v2	
Panel1		**Panel1**		Panel2		Panel2	
Panel2		**Panel3**			**Panel1**		**Panel1**
		Panel2				**Panel3**	
		*Add Panel3		*Move Panel1 into Panel2		*Panel3 relocated	

Renaming Elements

Case 1

Form Builder v1		Design Layout v1		Form Builder v2		Design Layout v2	
Panel1		Panel1		Panel1		Panel1	
E1	E4	E1	E4	**E1 -> E7**	E4	**E7**	E4
E2	**E5**	**E5**	E6	E2	E5	E2	E6
E3	E6	E2		E3	E6	E3	
		E3				**E5**	
		*Move E5 after E1		*Rename E1		*E5 relocated	

Case 2

Form Builder v1		Design Layout v1		Form Builder v2		Design Layout v2	

Panel1		Panel1		Panel1		Panel1	
E1	E4	E1	E4	**E1 -> E8**	E4	**E8**	E4
E2	E6	**E5**	E6	E2	E6	E2	E6
E3	E7	E2	E7	E3	E7	E3	E7
		E3				**E5**	
Panel2		Panel2		Panel2		Panel2	
	E5				**E5**		
		*Move E5 after E1		*Rename E1		*E5 relocated	

Deleting Elements

Form Builder v1		**Design Layout v1**		**Form Builder v2**		**Design Layout v2**	
Panel1		Panel1		Panel1		Panel1	
E1	E4	**E1**	E4	E2	E4	E2	E4
E2	**E5**	**E5**	E6	E3	**E5**	E3	E6
E3	E6	E2			E6	**E5**	
		E3					
		*Move E5 after E1		*Delete E1		*E5 relocated	

Browse Record Count

You can disable the global browse record count and instead enable the browse record count only for specified browses.

When a browse is run, the record count function retrieves and reports the total number of records in the browse, in advance of returning and displaying the actual records. Depending on the definition of the particular browse, this record count function can consume system resources and affect performance, delaying the display of the record data. With the global browse record count, the record count function is applied to every browse, which can affect overall system performance.

Previously, the browse record count was enabled or disabled globally, where the setting would apply to all browses being run. Now, you can enable it for only specified browses, tailoring it for use only with browse definitions that require a record count.

To configure the browse record count on a per-browse basis, in the `client-session.xml` configuration file, use the `<recordcount>` settings:

```
<!-- This section controls the record count done within browses -->
<recordcount>
<!-- Indicates if default is to skip record counts for browses -->
<defaultskip>false</defaultskip>
<!-- These browses will not be counted if default is to count -->
<!-- Format is csv list of browse ids, i.e. mg003,pp100,BCreditor.SelectCreditor -->
<skip></skip>
<!-- These browses will be counted if default is to skip. Same format as skip list -->
<count></count>
<!-- True will skip record count in all lookups. False will follow logic above -->
<lookupskip>true</lookupskip>
</recordcount>
```

When the global browse record count is disabled, the total record count number in the bottom-right corner of the display is replaced by “many”, such as “1-200 of many”. When you hover the cursor over “many”, a “Count Records” tooltip is displayed: by clicking “many”, you can select a “Count Records” option, which starts a record count. During the count, the display changes to “1-200 of [counting...]”. Once the count is complete, the record count number is then displayed, such as “1-200 of 23,510”.

Business Component Statuses

Introduction

This article describes the statuses of the business component and explains the following workflows:

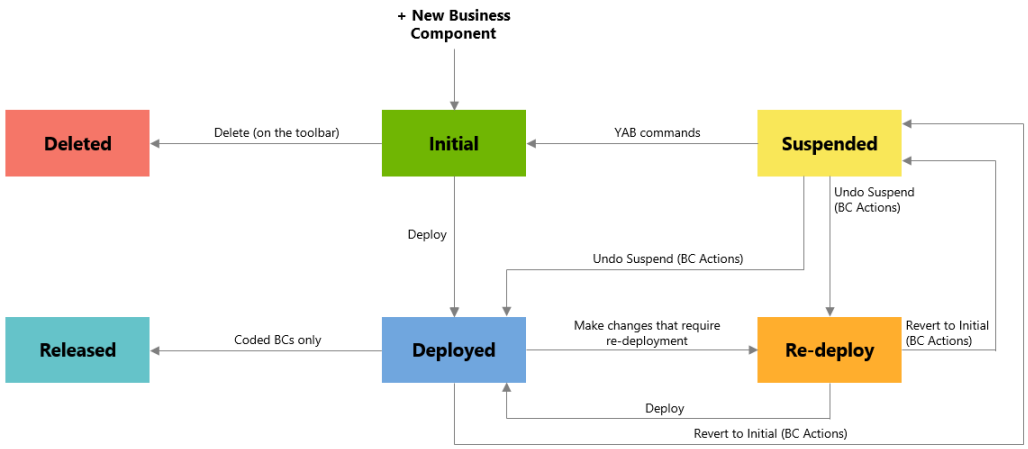
- Changing the structure of a previously defined business component by renaming or removing fields (including indexes)
- Removing a previously defined business component from the app

To support deleting a business component that is already deployed, a status field is introduced. To delete a business component, its status must be moved to Suspended (via Revert to Initial (BC Actions)), and then to Initial (via the YAB commands). Once in the Initial status, the business component can be deleted (via Delete on the toolbar).

Moving the business component back to the Initial status also allows updates to be made to the business component that cannot be made in the Deployed status. For example, changing the primary key. Here is a list of fields you can still edit when the business component is in the Deployed status.

Once a business component is released, it cannot be removed anymore or changed structurally.

Business Component Status Workflow



Status Description

Description of different statuses of a business component:

Status	Description	Notes
Initial	A status that the business component is in when created, but not yet gone through the initial deployment, or when the business component is reverted to Initial by running the YAB commands. This business component cannot be used yet in any runtime mechanism.	Everything can be changed. There is no physical table in the database that represents the data for this business component.
Sus	The business component gets into this status when you actually decide that this business component needs to be removed or fundamentally changed.	When a business component is suspended, the only thing that changes is the status. Nothing is removed yet when you suspend a business component.

pended	This business component cannot be used anymore in any runtime mechanism until it is deployed. In this status, the entity metadata for the current business component is fully disabled on the Web UI and cannot be modified.	
Deployed	In this status, the business component can be used at runtime.	This is actually the only status a business component can be in within a production environment.
Re-deploy	This is the status the business component gets in when certain properties of the business component are changed, which require a manual re-deployment to ensure everything is generated correctly for the business component to be available at runtime.	This status is set automatically when you make changes and click Save.
Released	This status indicates that the business component is released and is part of an officially released app package. Probability is high that it is already in use in a customer environment.	Currently, this status is for the coded business components only. Coded business components always have the Released status. This means that they cannot be changed in the business component builder. There is no way yet to set a platform business component to Released.

Business Component Status Changes

Possible scenarios of business component status changes:

Status	**Next Possible Status**	**Definition of Situation**
Initial	Deployed	This is the initial definition of the business component. You define the business component and after reviewing all field definitions, as well as form and view definitions, decide to deploy the business component for testing. Deploying the business component automatically ensures the right table and fields are in the database.
Initial	<removed>	You started to define a business component but realized that this is the wrong one (wrong name, wrong structure, in the wrong app, and so on). So you decide just to delete the business component from the business component builder (via Delete on the toolbar).
Deployed	Re-deploy	After testing of the business component, you must make further changes to the business component, mostly adding fields, adding indexes, changing labels, adding relationships, adding formula fields, and so on. In many cases, this results in the fact that the business component definition now deviates from other artifacts that are generated and recorded in the system. At that moment, the system puts the business component in the Re-deploy status, and the business component becomes unusable at runtime.
Depl	Sus	You tested the environment and found that there were fundamental issues with the structure of the business component. A change that is normally not possible via the business component builder or removal of the business component is needed. If this is the case, the first step you need to take is to suspend the business

oyed	pended	component. During the time the business component is in the Suspended status, the business component cannot be modified in the business component builder, nor it is usable at runtime. Next, you can either Undo Suspend (via BC Actions) or run the YAB commands that will remove the schema and return the business component to the Initial status.
Deployed	Released	You tested the business component in different scenarios and all processes in which the business component takes part. Then, you decide to create a release package for the app. At that moment, it is a good idea to release all business components that are included in the app. A business component that has ever been in the Released status can never be suspended anymore, as that would allow backward compatibility breaking changes.
Re-deploy	Deployed	After making changes to a business component that caused the business component to become re-deployed, you decide to deploy it again.
Re-deploy	Suspended	You tested the environment and found that there were fundamental issues with the structure of the business component. A change that is normally not possible via the business component builder or removal of the business component is needed. If this is the case, the first step you need to take is to suspend the business component. During the time the business component is in the Suspended status, the business component cannot be modified in the business component builder, nor it is usable at runtime. Next, you can either Undo Suspend (via BC Actions) or run the YAB commands that will remove the schema and return the business component to the Initial status.
Suspended	Initial	You make the business component available again for editing in the business component builder by running the YAB commands. These commands drop the table associated with the business component (including the data) and put the business component in the Initial status again. Then, you can make any changes to the structure of the business component or may decide to remove the business component.
Suspended	Deployed/Re-deploy	You can Undo Suspend if you suspended a business component by mistake. The business component will revert to its previous status, Deployed or Re-deploy accordingly.

Deleting a Business Component

A business component can be deleted when it is in the Initial status. These steps involve actions on the Web UI and a YAB console command, as follows:

1. On the Business Component's Actions drop-down, choose Revert to Initial.
   The business component is in the Suspended status now.
2. Next, use the following YAB commands in this order. Pay attention that these commands drop the table associated with the business component (including the data).
   ```
   yab stop
   yab database-extension-obsolete-schema
   yab start
   ```
3. The business component is in the Initial status now.
4. Finally, on the Business Component toolbar, the Delete button is now available. To delete the business component, click the Delete button.

Editing a Business Component (Reverting a Business Component to Initial)

When the business component is reverted to the Initial status, it allows updates to be made to the business component that cannot be made in the Deployed status. These steps involve actions on the Web UI and a YAB console command, as follows:

1. On the Business Component's Actions drop-down, choose Revert to Initial.
   The business component is in the Suspended status now.
2. Next, use the following YAB commands in this order. Pay attention that these commands drop the table associated with the business component (including the data). You can export the data before running these YAB commands, and then import the data if needed.
   ```
   yab stop
   yab database-extension-obsolete-schema
   yab start
   ```
3. Now, the business component is in the Initial status and is available again for editing in the business component builder.

List of Fields You Can Edit in Deployed Status

This article describes which properties of the business component and fields you can change when the business component is in the Deployed status.

Main Panel

UI Element	Possible Property Changes	Notes
Business Component Label	Yes	
Description	Yes	
Scope	Yes	
Business Document	Yes	
Approvals	Yes	

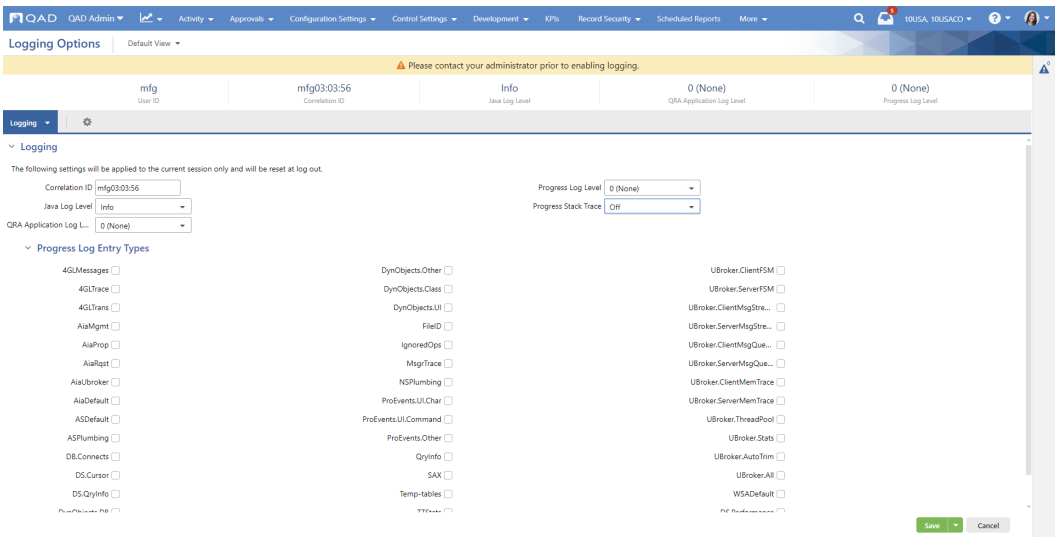
Fields Panel

UI Element	Possible Property Changes	Notes
Field Label	Yes	
Lookup	Yes	You can edit it in the Field Details pop-up window.
Format	Yes	It is disabled for the Character and URL data type.
Default Value	Yes	It is always disabled for primary fields.
Formula	Yes	You can edit the formula definition, not the Formula checkbox.
Minimum Value	Yes	
Maximum Value	Yes	
Required	Yes	It is always set to Yes (Selected) for primary fields and cannot be changed. For other fields, it is editable.
Read Only	Yes	It is always set to False (Not Selected) for primary fields and cannot be changed. For other fields, it is editable.
Hidden from Activity Tracking	Yes	
Sortable	Yes	It is always set to Yes (Selected) for primary fields and cannot be changed. For other fields, it is editable.

Logging Options view

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.

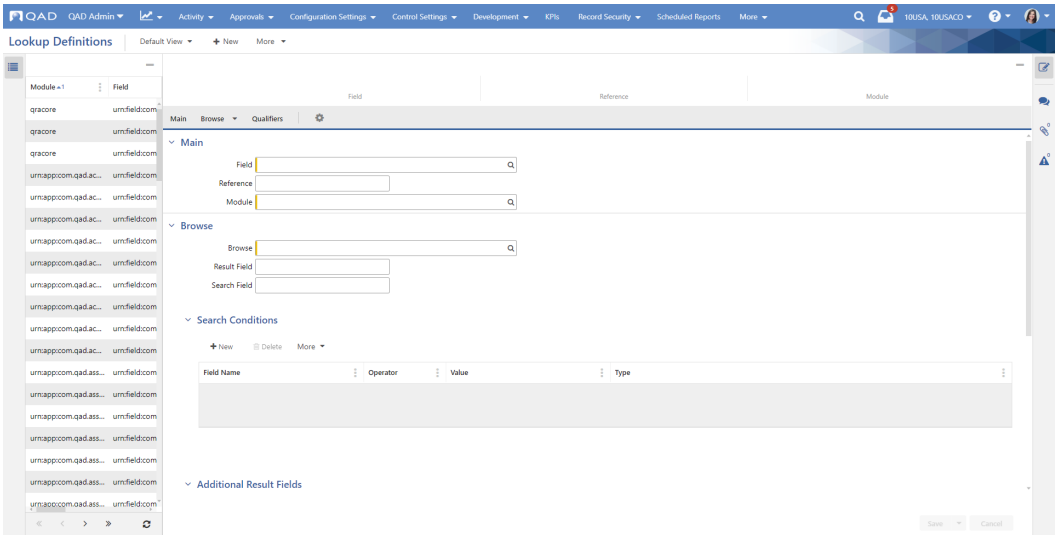
You can define lookups for a business component field or at the level of a view field. Lookups defined for a business component field can then be used by all the view fields that are based on the business component field. Lookups defined for a view field can only be used by the field in that particular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

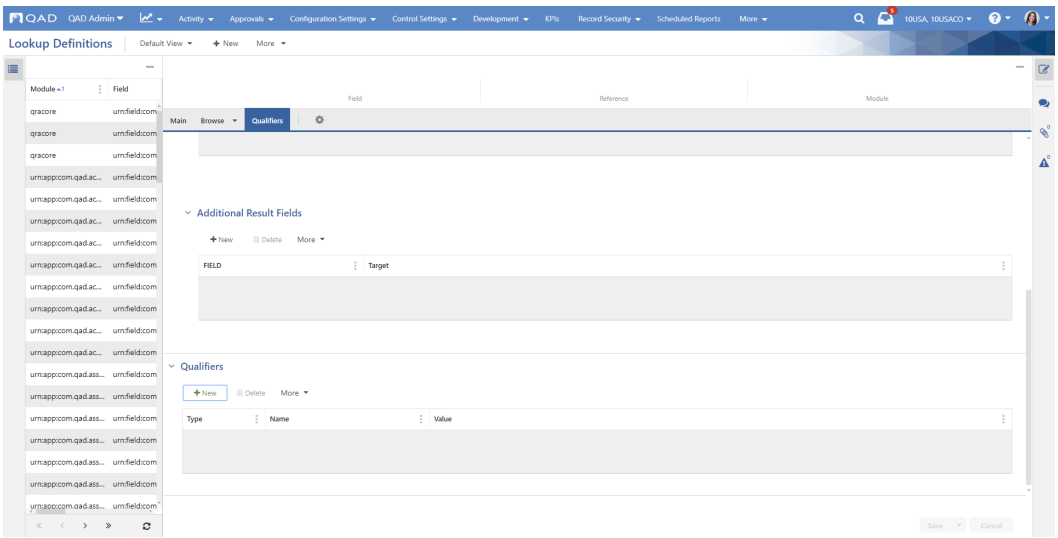
When you open Lookup Definitions, you first get a listing of the settings for the currently defined lookups, including:

- 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 OOABL Business Logic

Table of Contents:

Introduction

This section contains detailed information that can be used by an extension developer to extend the OOABL business logic flows for business components. To start coding an extension, you need to know the **method**, the **Business Component**, and the **App**.

There is a difference between what is called **Coded business components** and **Platform business components**. The way to extend is almost the same, there are slight differences though. There are exercises that detail both of them.

Extensions on the business logic are typically required for:

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

We use the case to extend Countries as described in Example 4: Extend Countries BC with an embedded BC (Capital, Population, Currency Code) and add OOABL validation for Currency Code

Note: Also, the server-side scripting can be considered as alternative way to extend the business logic.

1- Channel Islands business component patterns

Most of the business components in the system can be accessed through a hybrid browse on the menu. The system uses **fixed patterns** to work with the data. Both coded and platform 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, and run on the appserver.

2- Channel Islands business component hooks

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.
Methods that are not exposed in the interface can not be extended. To know what is exposed in the interface you can look at the **HTML documentation**. For this, you need to know the App and the Business Component.
The HTML documentation shows you the names of the entities, methods, temporary tables, variables that can be used.

3- How to find App and Business Component

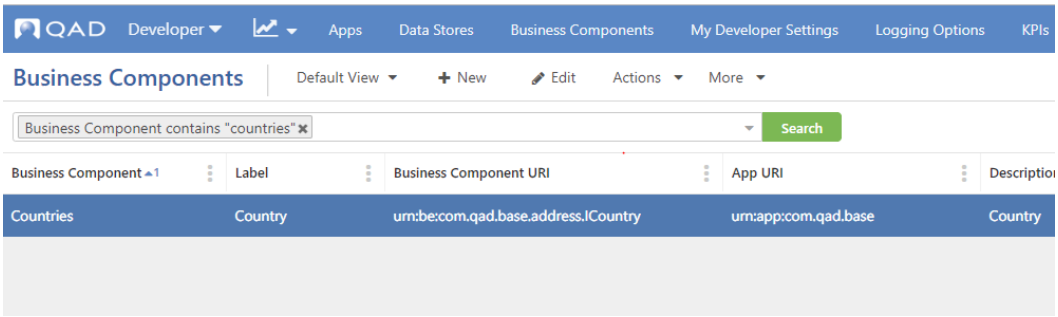
As indicated in the introduction we need to determine the exact implementation of the function we want to extend. Besides the methodname, we need to know the

- Business Component
- App

A way to get the App and Business Component you can do the following:

1. Run WebUI.
2. Select **Business Components** from the menu search.
3. Search for the Business Component you want to extend. You can use different filters and operators to narrow down the selection.
   The browse will show you the BC URI and the App URI.

Example for Countries:



You will find the URI of the Business Component and the URI of the app:

- urn:be:com.qad.base.address.ICountry, so the Business Component is 'ICountry'
- urn:app:com.qad.base, so the appName is 'base'

However, when the business component is an extension on its own (called **platform business component**), the business component to extend is **com.qad.qra.be.IVirtualBusinessEntity.**

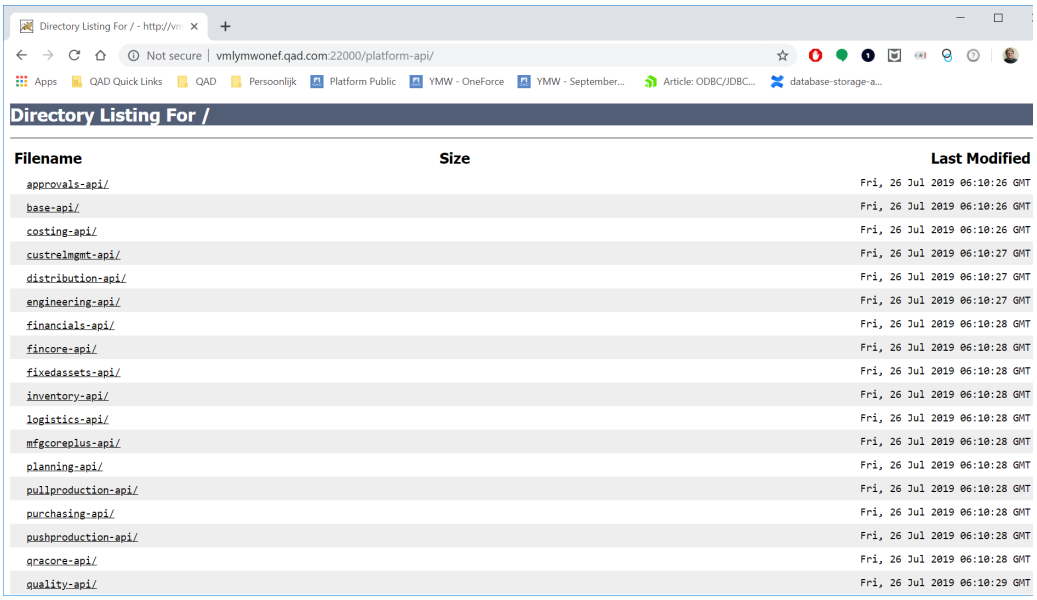
4- Finding and using HTML Documentation

The HTML documentation pages can be retrieved as follows:

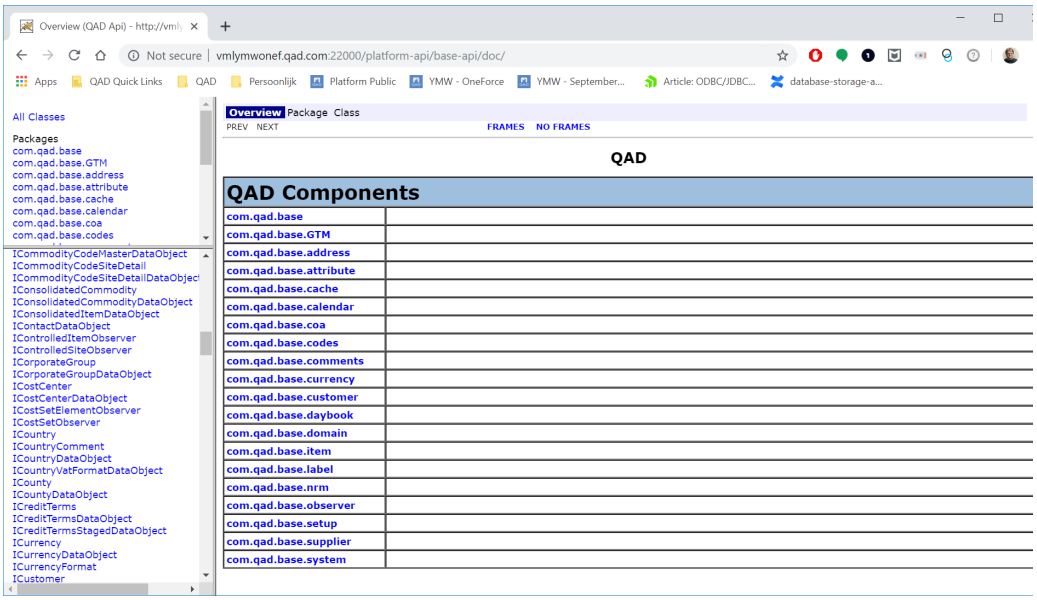
- Run yab env-info in your environment and you'll see where it is located.

```
[mfg@vmlymwonef systest]$ yab env-info

Client
                QAD Home          http://vmlymwonef.qad.com:22000/qadhome
                .NET UI           http://vmlymwonef.qad.com:22000/qadui
                QAD Web UI        https://vmlymwonef.qad.com:22011/qad-central
                QAD Web UI Proxy  https://vmlymwonef.qad.com/clouderp
Platform
                Platform API      http://vmlymwonef.qad.com:22000/platform-api
QXtend
                Inbound           http://vmlymwonef.qad.com:22178/qxi
                Outbound          http://vmlymwonef.qad.com:22178/qxo

* To print more detailed information use (-more).
```

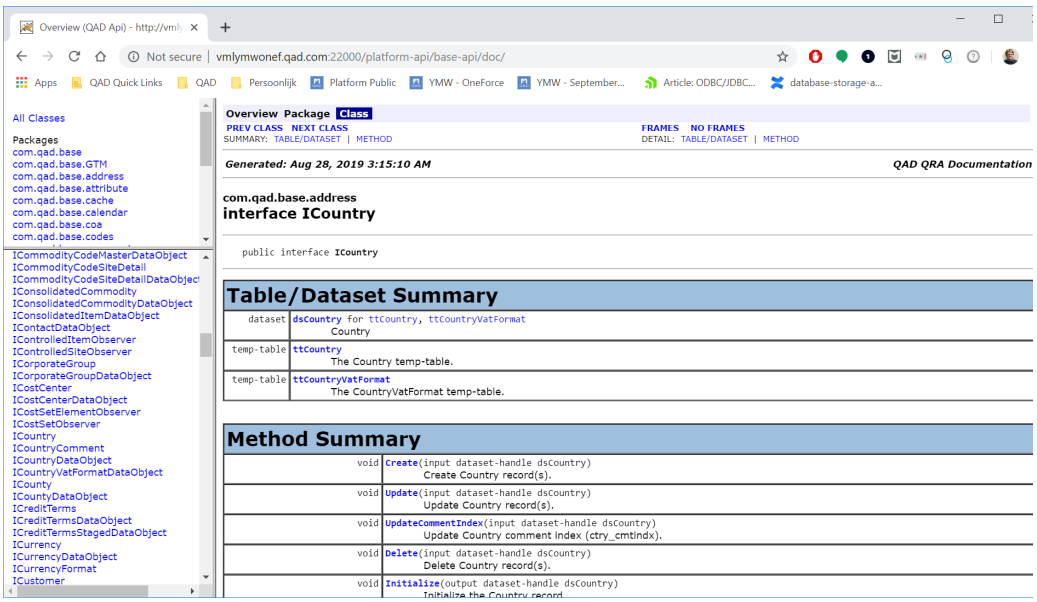
- Opening that URL will show you a list of all apps installed in your system.



- Select the App that you found in the previous step.
  If you click, for instance, on base-api and then on doc, you'll see an overview of everything included in the base app.



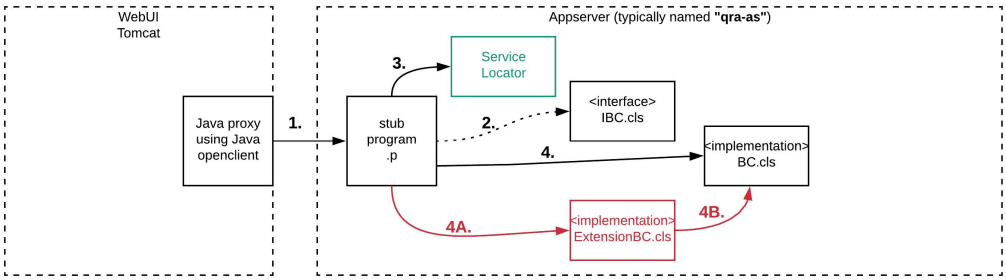
- Select the Business Component you found in the previous step.
  If you then click on ICountry for example, you'll get an overview of all methods that are exposed in this class.



5- 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: (stub.p ServiceLocator implementationBC.cls)

1. **Java service**
   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. **Stub programs**
   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. **Service Locator**
   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. **Implementation code**
   Using the instance the stub got from the Service Locator, the right method is called and all lower level implementation code gets executed. In this case this is the **<implementation>BC.cls**

Extended flow: (stub.p ServiceLocator implementationExtensionBC.cls implementationBC.cls)

3 **Service Locator**
The ServiceLocator returns an alternative implementation to the stub program. In this case it is **<implementation>ExtensionBC.cls**. This implementation class 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.

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

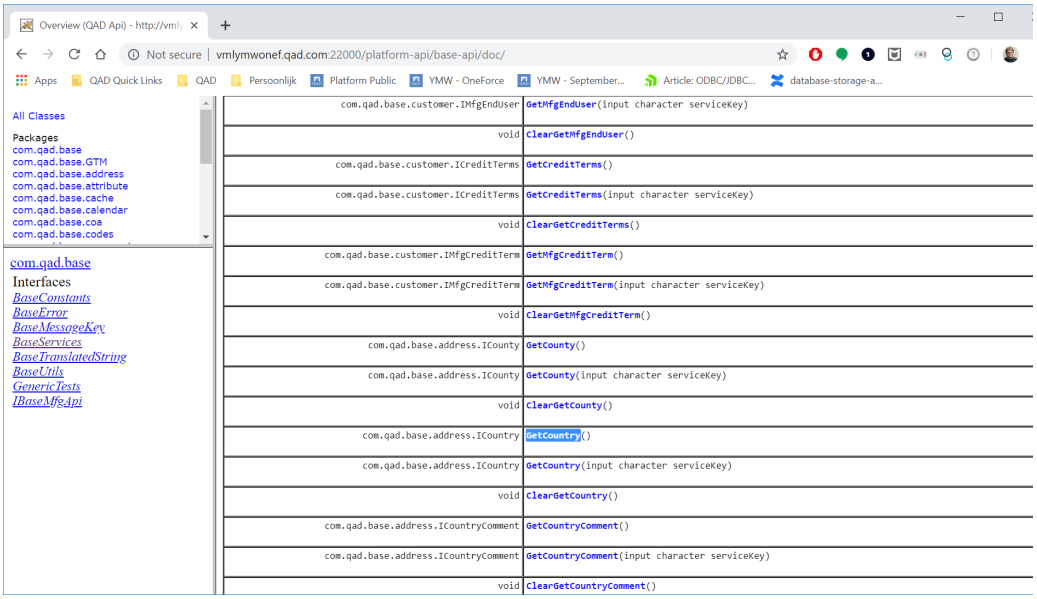
6- How to get an instance of a service you want to work with

'Services' class and 'Get' methods

Each app has a class that exposes a way to get to the services in that app. It can be found in the HTML documentation at the top namespace of the app and is named <AppName>Services. It has methods defined for each Business Component that can be used to return an instance of that Business Component.

Our example:

- open HTML documentation and go to the base app
- if you click on com.qad.base under the Packages list, you'll see a class named BaseServices.
- Then you'll see a GetCountry() method listed, which will return you an instance of Country.



- You can use this method in your code.
- The code to retrieve that instance and use it would be:

Example

```
define variable countryService as com.qad.base.address.ICountry no-undo.

// Get an instance of country
assign countryService = com.qad.base.BaseServices:GetCountry().

// Do something if the country is not active
if not countryService:IsActiveCountry(input "US")
then do:
    ...
end.
```

7- How to find the Error object

To find out if an app has an error object, you can use the same approach as above to get an instance of a service. On the top namespace of an app, there is a calls <AppName>Error.

In our example, you will find the BaseError object. You might need this when developing your extension.

8- How to do extra validations

You can add extra validations to your code like this:

```
if <error condition>
then do:
    <create error message>
        <set error context>
        <add error to list of validation errors>
end.
```

Example:

```
define variable bufCrmAccount    as handle  no-undo.
define variable bufCrmAccountExt as handle  no-undo.
define variable valMsgs          as List    no-undo.
define variable msg              as Message no-undo.

assign valMsgs          = new List()
       bufCrmAccount    = dsCrmAccount:get-buffer-handle("ttCrmAccount")
       bufCrmAccountExt = dsCrmAccount:get-buffer-handle("ttCrmAccountExt").
```

```
bufCrmAccount:find-first().
bufCrmAccountExt:find-first().

/* Extension field EurAmount is mandatory when standard field CurrencyCode is EUR */
if bufCrmAccount::CurrencyCode = "EUR" and
  (bufCrmAccountExt::EurAmount = ?     or
   bufCrmAccountExt::EurAmount = 0)
then do:
    /* First parameter is a message number you can choose yourself. */
        /* Second parameter is the error message.                      */
    assign msg = new Message(666, "Field is mandatory when Currency Code is EUR.").

        /* Set the context to the field in error. This will make sure the UI can draw the red
border around the correct field. */
    msg:SetContext(bufCrmAccountExt:buffer-field("EurAmount")).

    /* Add the message to a list of messages you want to be shown in the error viewer on the UI.
*/
    valMsgs:Add(msg).
end.

/* Do any more validations you need. */
...

/* If you do have validation errors, stop processing and throw
them.                                          */
/* Since this was a customization for the custrelmgmt app, I throw a CustRelMgmt
error.                          */
/* Most (if not all) apps have their error object. If an app doesn't have one, you can use the
QraError object. */
if not valMsgs:IsEmpty()
then undo, throw new CustRelMgmtError(valMsgs).
```

There is no support yet to add your own translatable strings or messages.

9- When to add code before and when after the standard implementation

In the section Executing the extension code, we mentioned you can insert code to run before or after the standard implementation.

Typical examples of code to add before the standard implementation:

- Doing extra validation.
- Changing the data before it is stored in the database.

Typical examples of code to add after the standard implementation:

- Doing extra updates to the database that are not part of the standard flow.

10- 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 business component: Example 4: Extend Countries BC with an embedded BC (Capital, Population, Currency Code) and add OOABL validation for Currency Code
- Extend a platform business component - delete an instance: Add OOABL validation to check if a tax category is still referenced when deleting (using IVirtualBusinessEntity)

11- Using Visual Studio Code

One of the steps you noticed in the examples above is, exporting your app to the file system. One of the ways to create /edit your custom code, is using vi, but that of course is not very user-friendly. You can use Visual Studio Code with the following extension installed:

Name: OpenEdge ABL
Id: chriscamicas.openedge-abl
Description: OpenEdge ABL language support for VS Code.
Version: 1.1.4
Publisher: Camicas Christophe
VS Marketplace Link: https://marketplace.visualstudio.com/items?itemName=chriscamicas.openedge-abl

Follow these steps to use Visual Studio Code to do your OOABL extension development.

- Add the above extension to Visual Studio Code.
- Map your linux vm to some windows drive on the machine you're going to use Visual Studio Code.
- Add a file named .openedge.json to the export folder (don't forget the period at the beginning of the filename). The content should look similar to this:

```
{
    "proPath": [
        "${workspaceFolder}\\src\\impl",
        "${workspaceFolder}\\lib\\custrelmgmt-api\\src",
        "${workspaceFolder}\\lib\\qracore-api\\src"
    ],
    "proPathMode": "prepend"
}
```

You need your src\impl folder in there and one entry for each dependency. The proPathMode entry means these folders will be added at the beginning of the standard propath.

You can now start by adding folders and files as described in the examples.

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

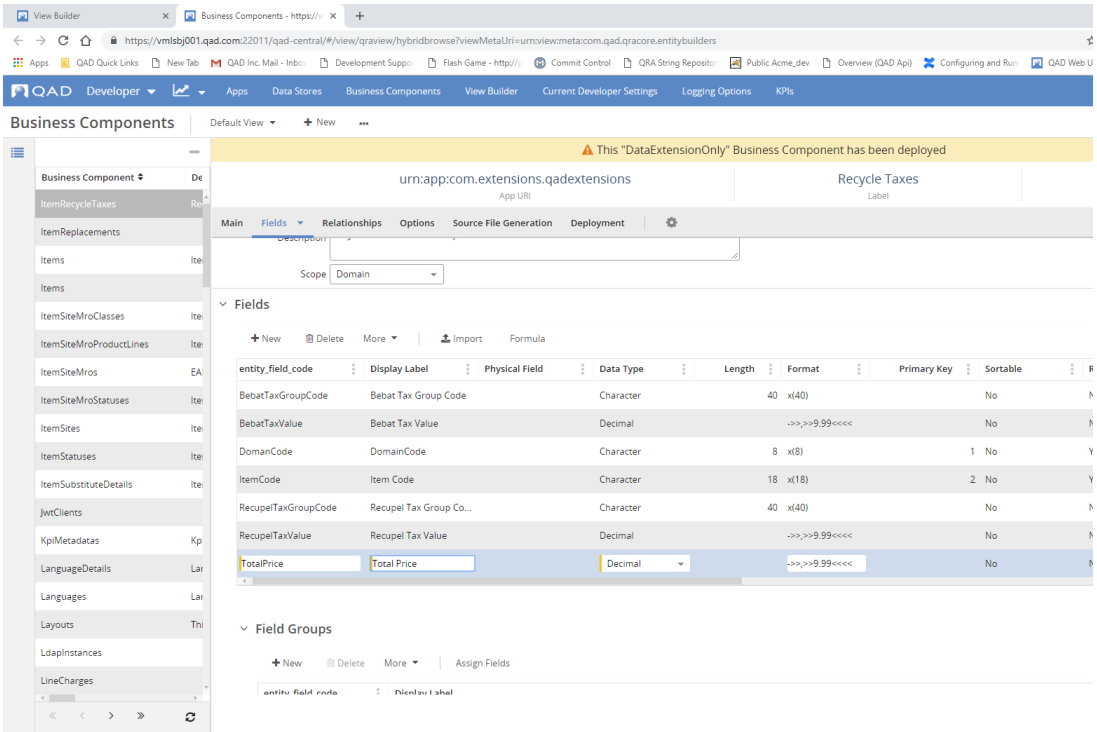
Formulas

Introduction

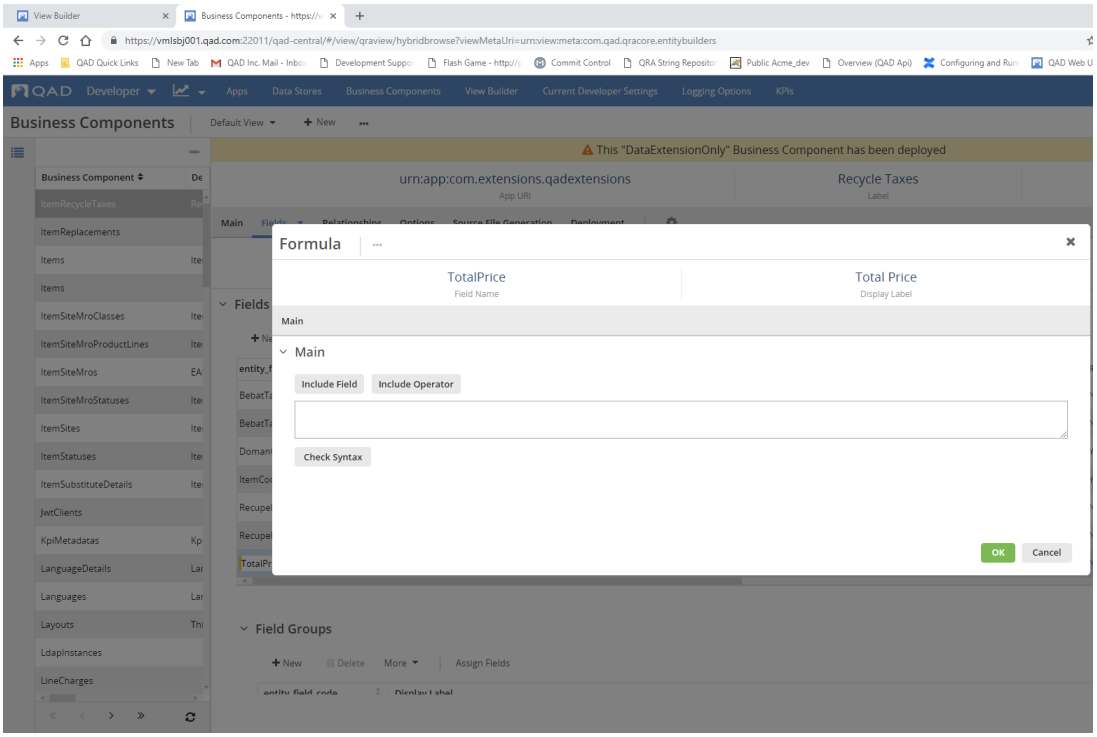
The business component builder provides the capability to add fields that are calculations based on data from other fields and records. We call these fields "formula fields" (see BC builder - formulas). This page describes how to extend a BC with a formula field.

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:

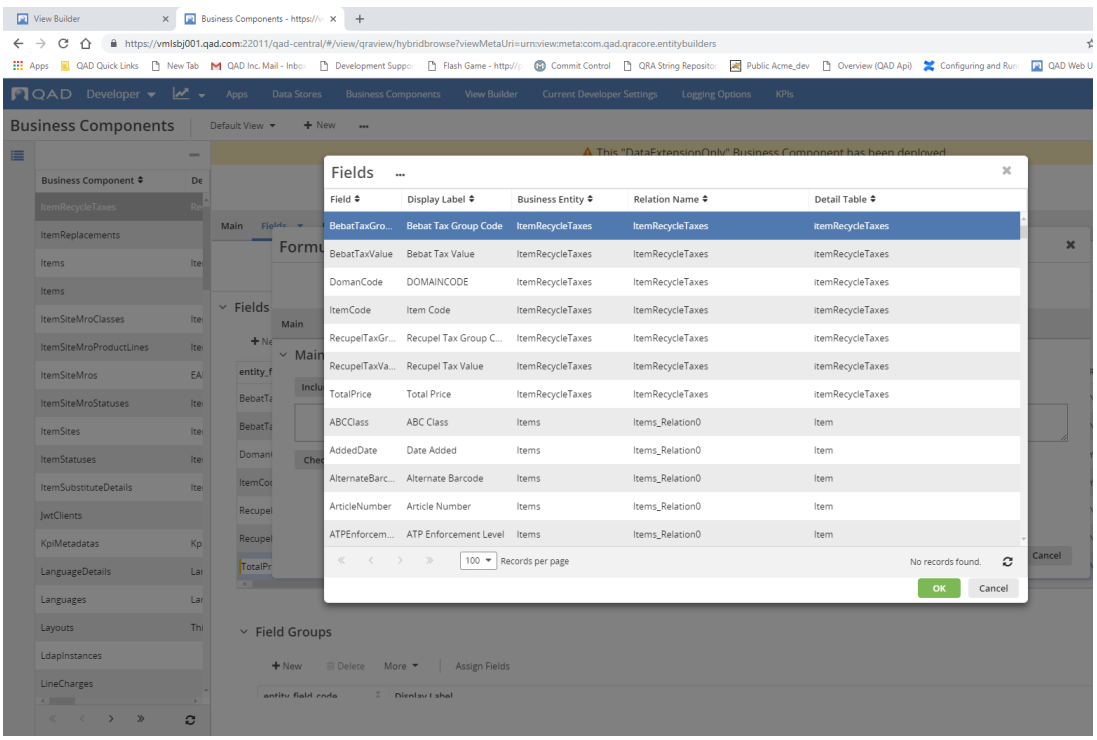


Note that you can do this on an already deployed Business Component. After adding the field press on the formula button on top of the grid:

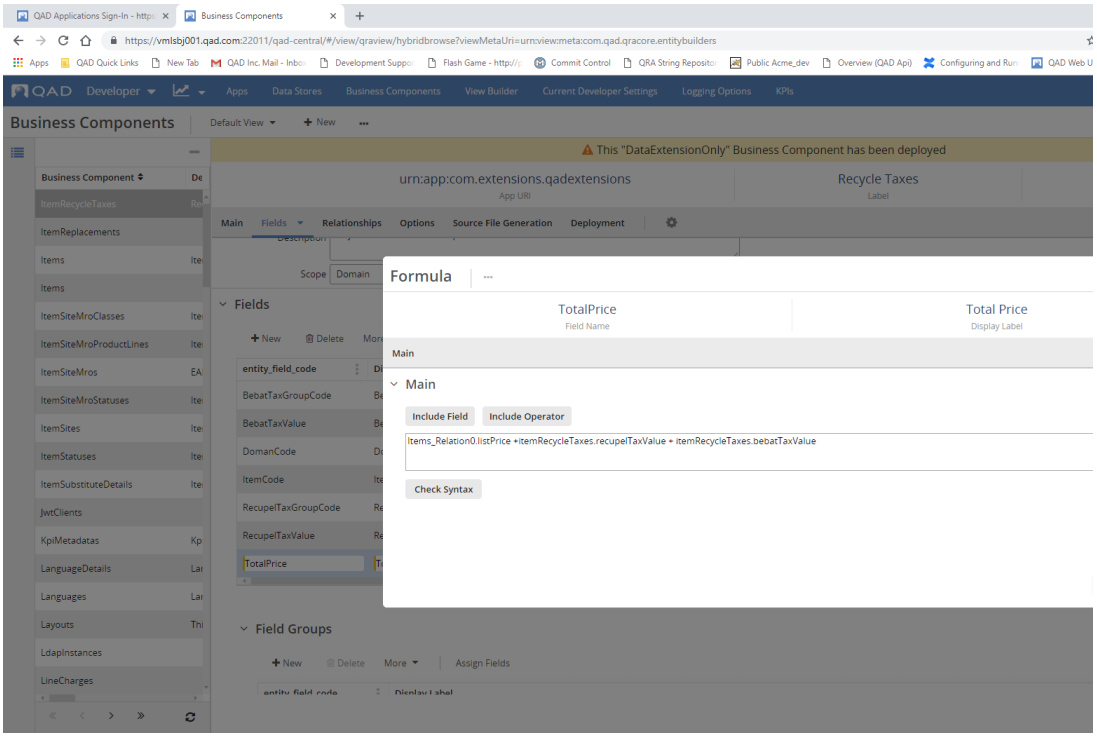


On the above screen you can start adding fields and operators to build a formula. You can also type the field names and operators without selecting them. The check syntax button always needs to be pressed before you are able to save the formula by pressing the ok button.

When selecting a field, note that you can take fields from the BC itself, but also from the child-parent relationships:



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 relationship:



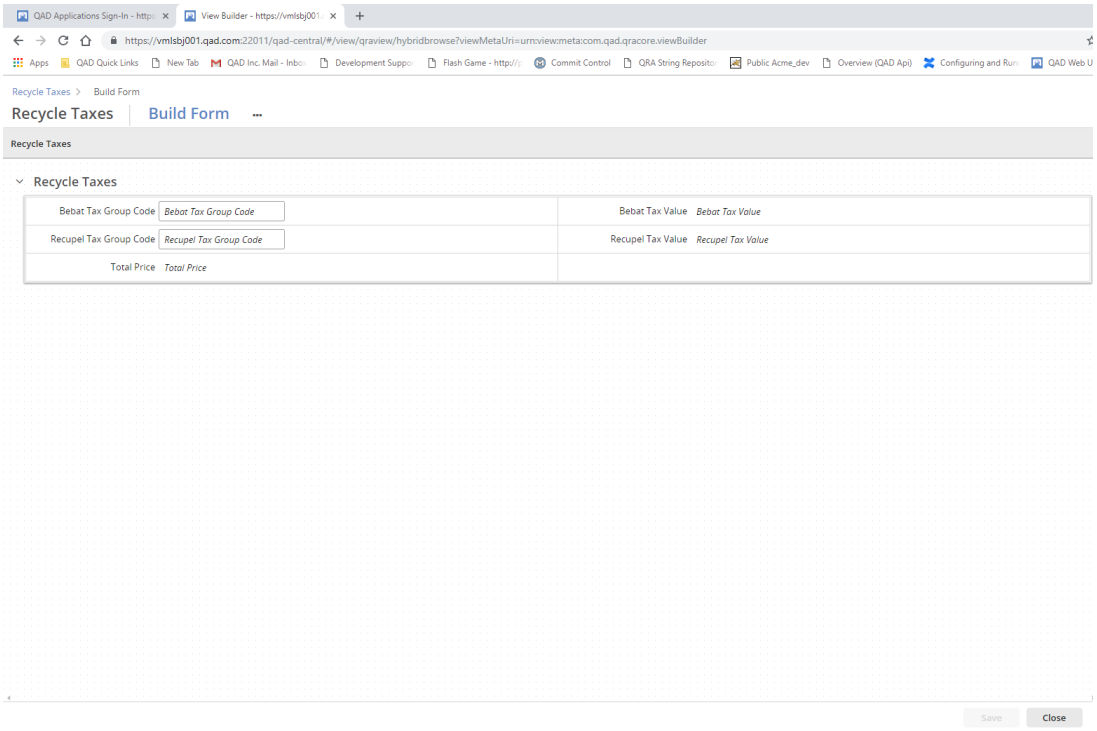
Aggregation functions

When the business component has a One-to-many child-parent 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 One-to-many relationship. However, it is important 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 metadata is cached and only reloaded on page refresh).

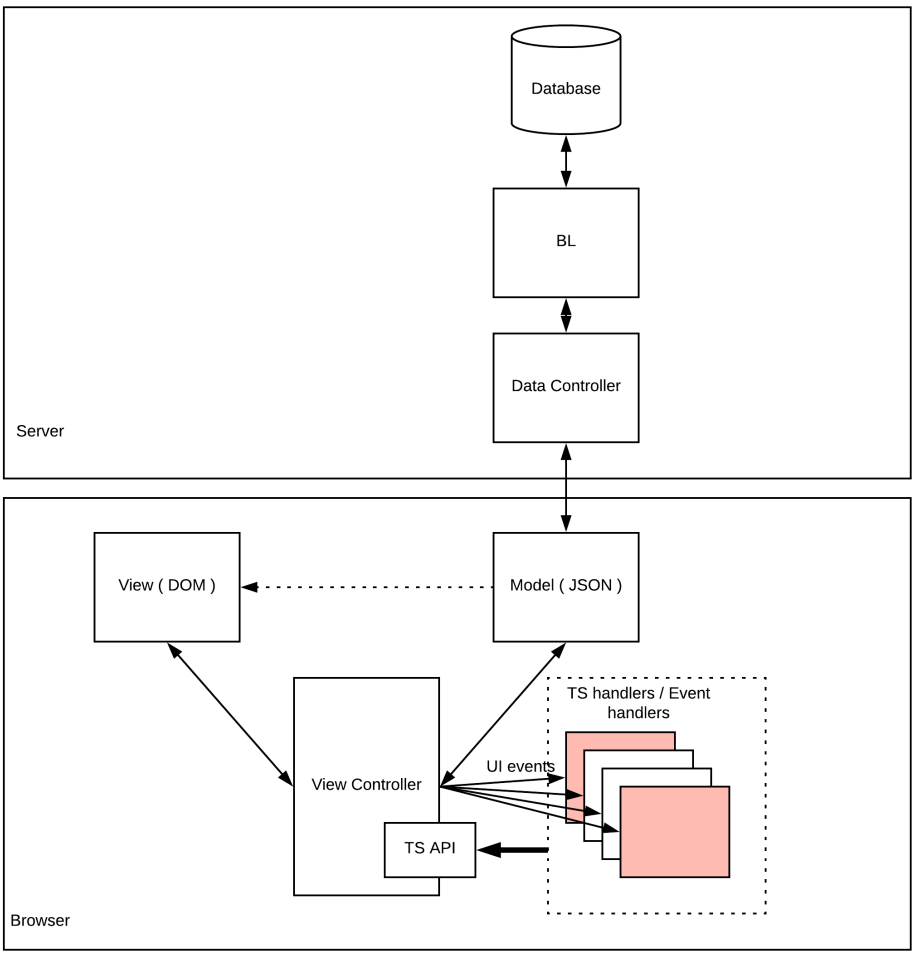
UI Event Handlers

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 platformbusiness 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 platform 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 platform 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.

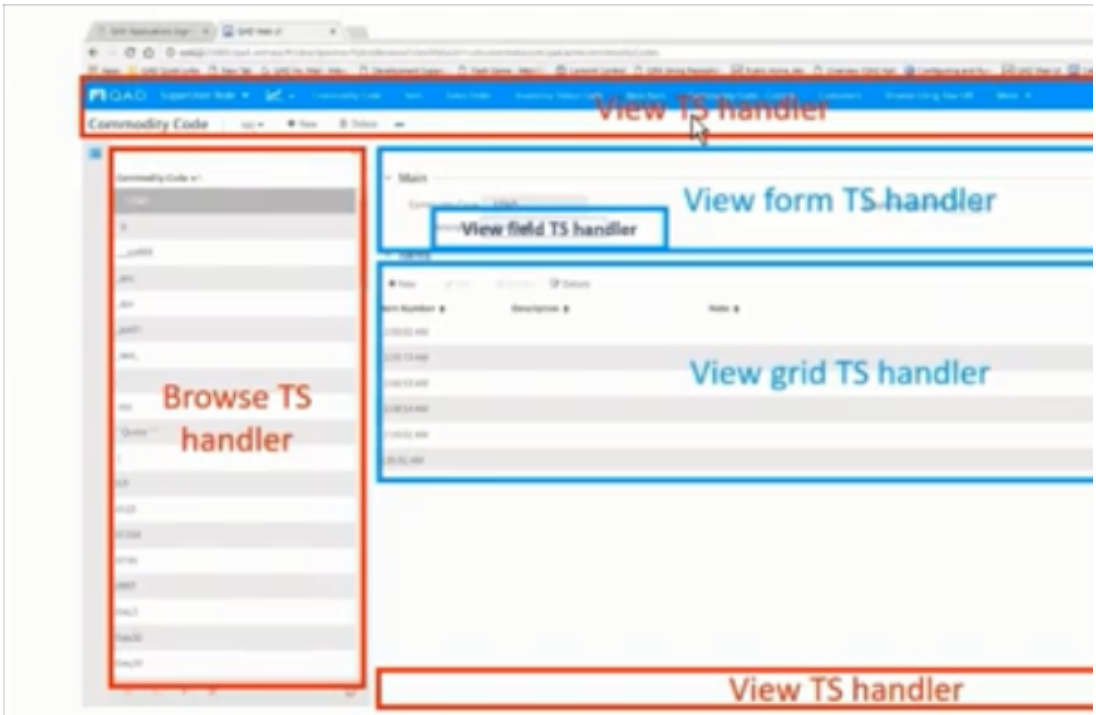
So, there are three types of Event Handlers:

1. Primary – primary event handler for corresponding business component. Can be created only for platform 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.

aSince 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 :

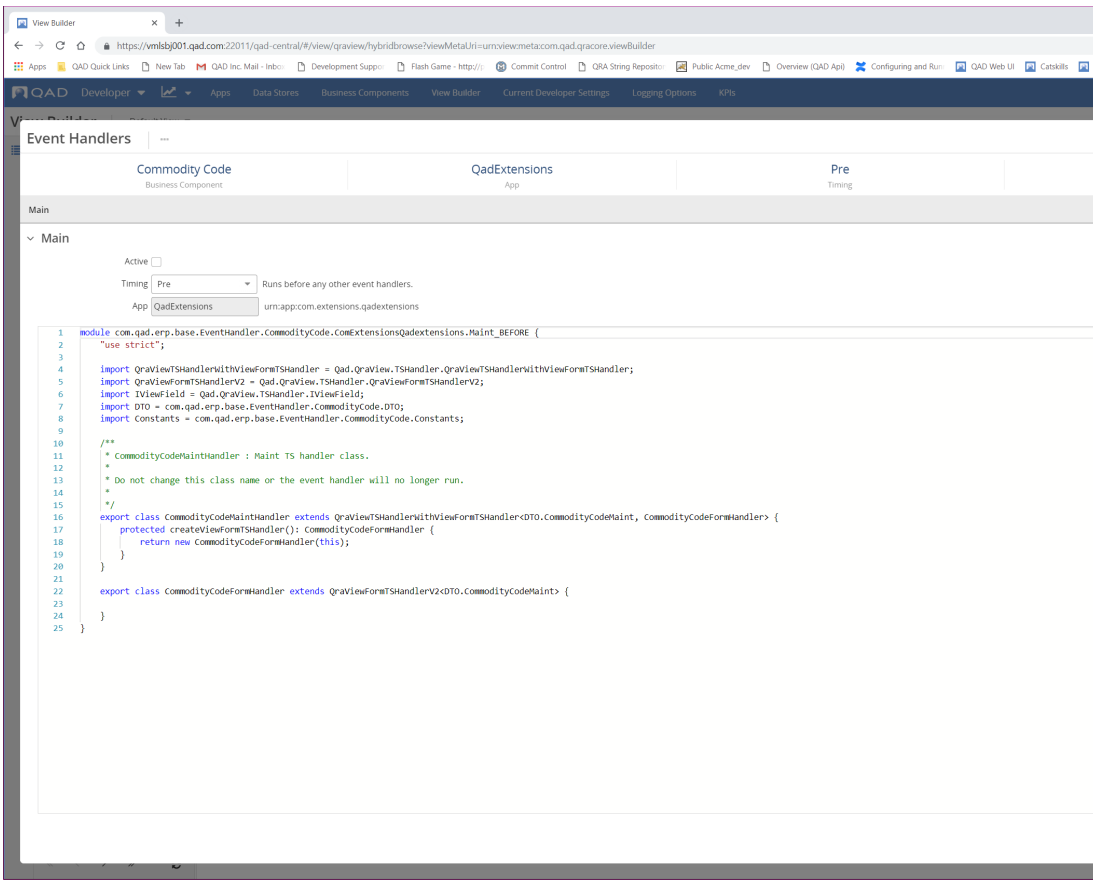
1. View TS handler : This is the main TS handler, and also the main UI event handler. This type is always needed as a starting point for event handlers. It can act on several more general UI events.

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).

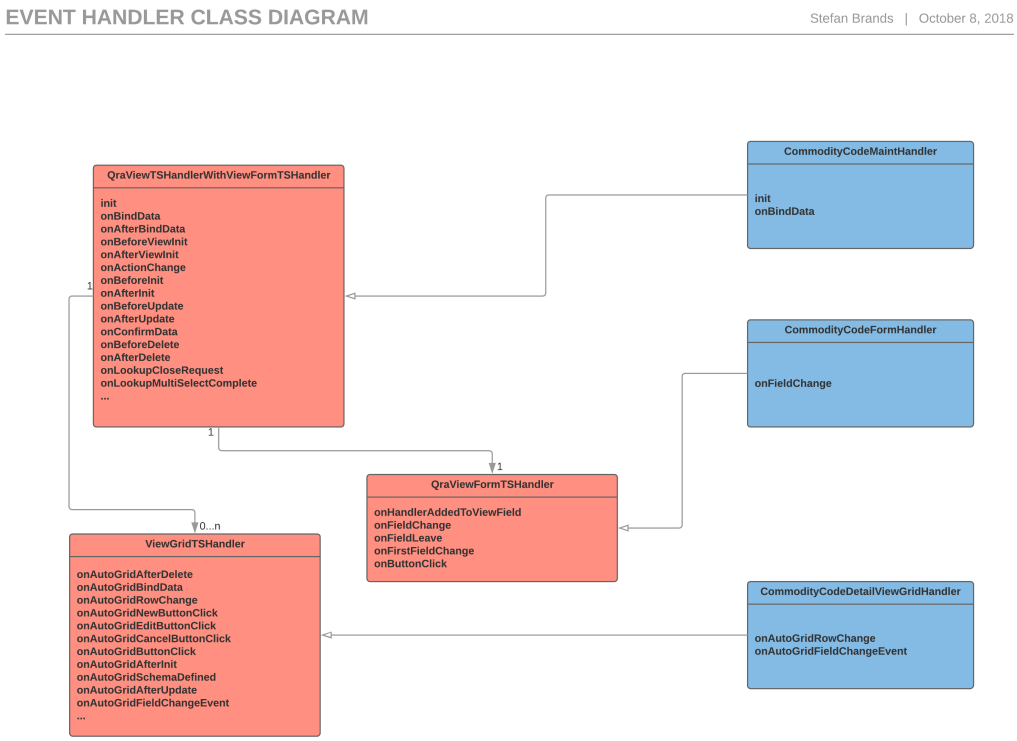
Why base classes ?

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

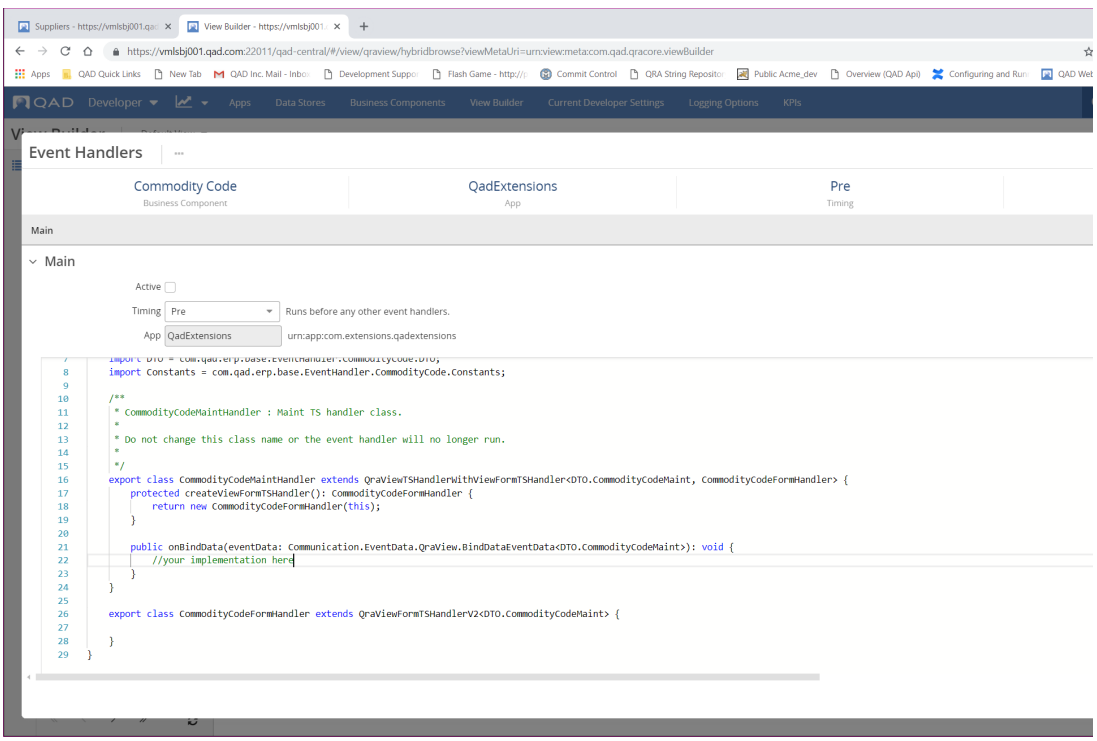
- They implement an interface that is called by the controller when a UI event is fired. All you need to do is write code that needs to run when an event happens is to override a base method, and add your own code.

- 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):



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:

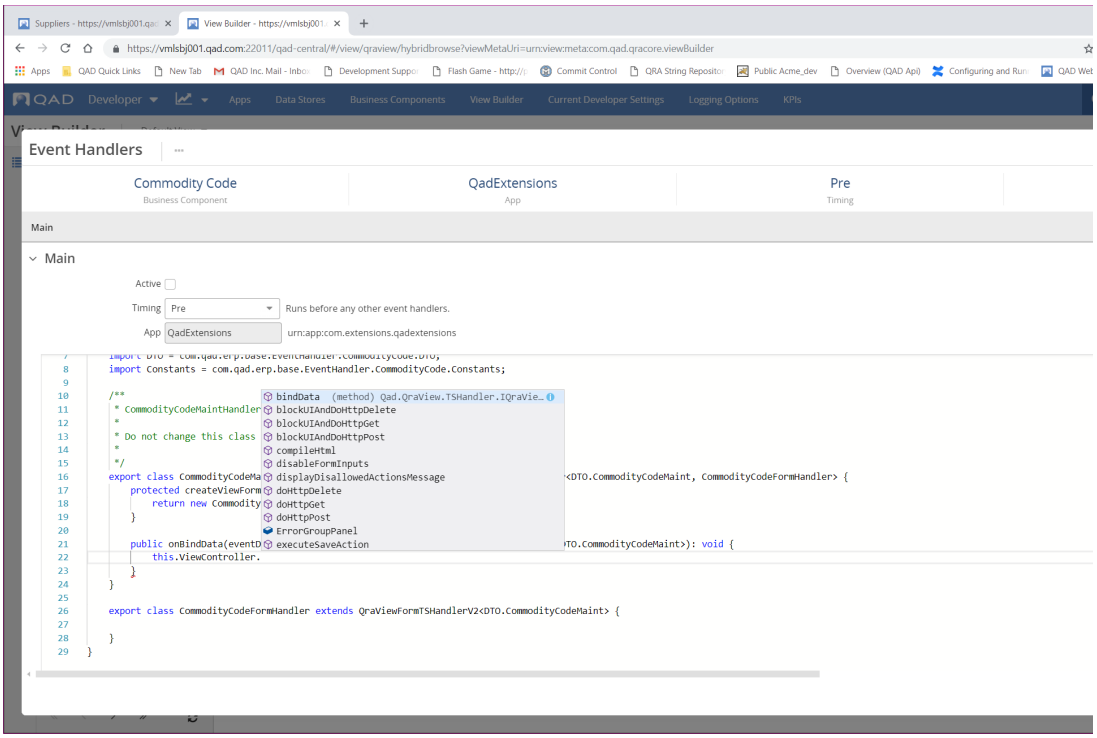


In order to add an event handler for a grid, a few lines of code need to be added. More detail can be found here : Handling grid events

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 :

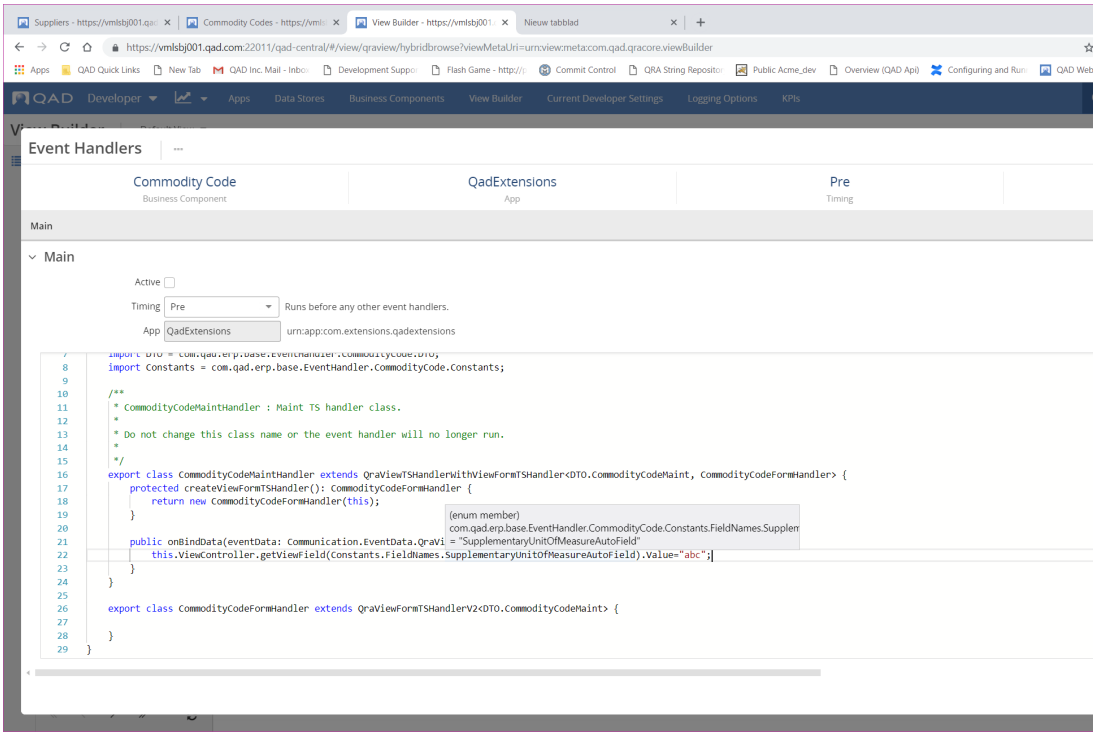
- getViewGrid
- getViewLabel
- getViewButton
- getViewField

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 :

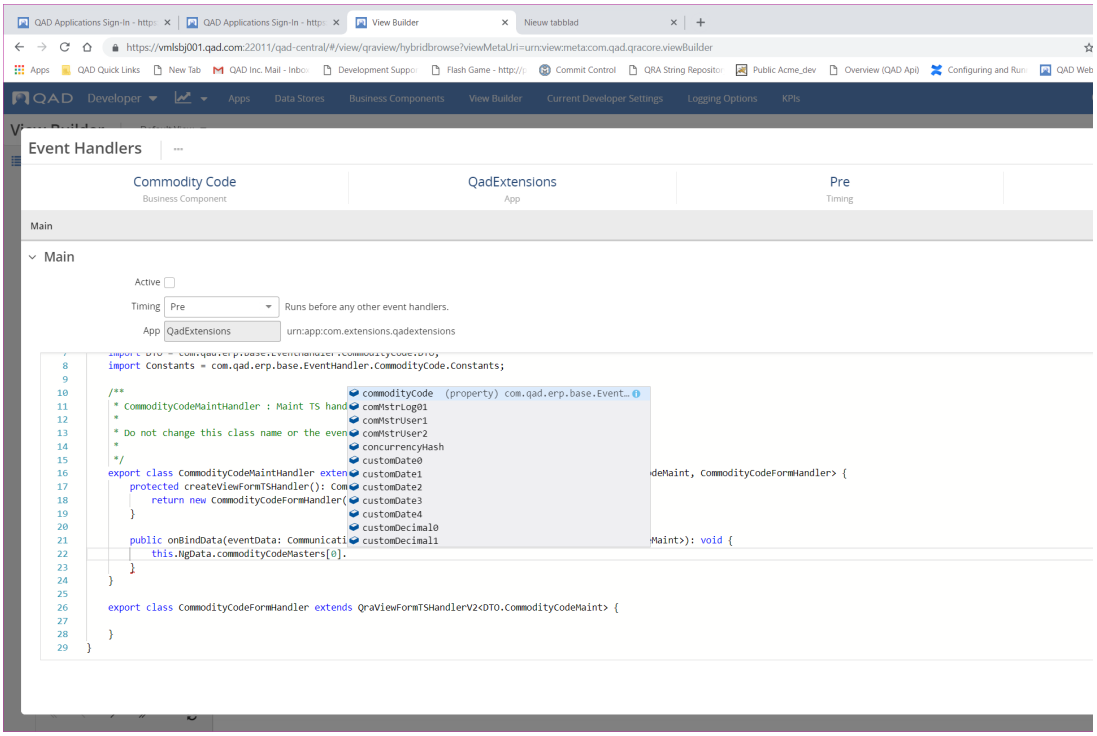


Note that we use the constant here, but we could have used the id in string too : this.ViewController.getViewField ("SupplementaryUnitOfMeasureAutoField"). ... .

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.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**: 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.
- The config/module-config.xml file will not be overwritten, but the ModuleInfo values will be updated while the rest of the file content is left as is.
- 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 your 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.

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 an 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:

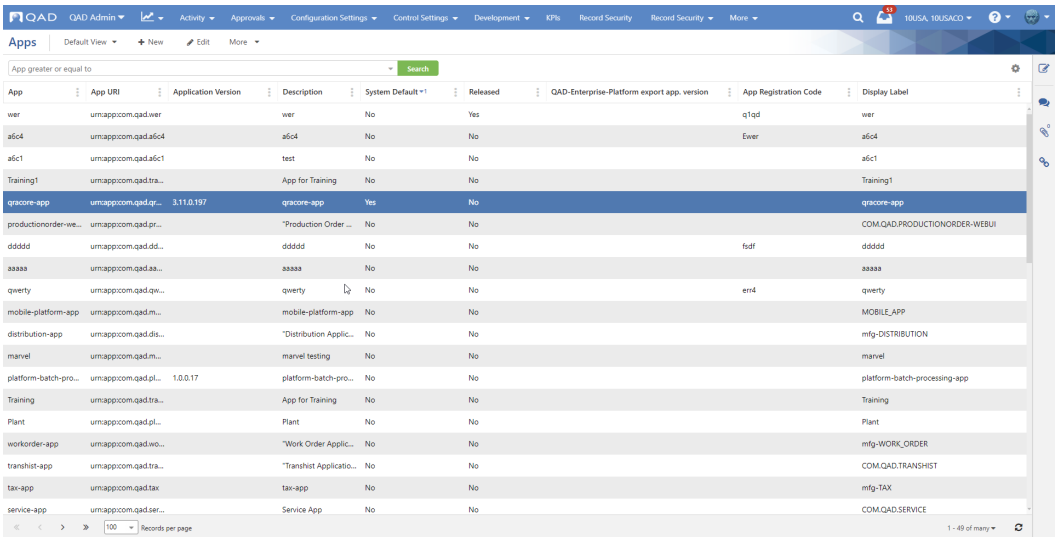
```
> yab update
```

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".

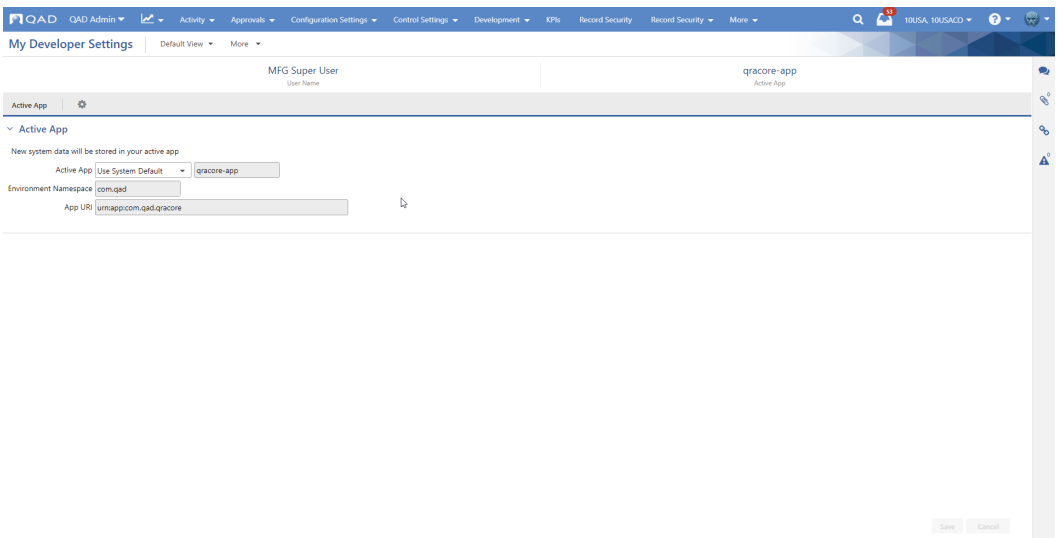


In this example, the system has default App with name "qracore-app" and with App URI "urn:app:com.qad.qracore".

App URI consists of environment namespace "com.qad" and the name of App "qracore". Having predefined environment namespace, all the further new Apps will be created in this environment namespace "com.qad".

2- View the Active App

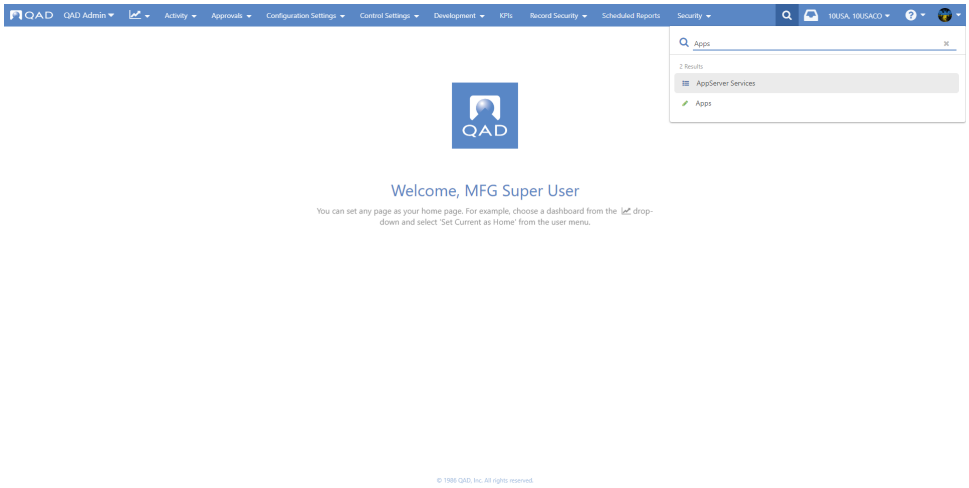
By default, the Active App in the system is set up as the Default App. This can be seen in the **My Developer Settings** for the current user.



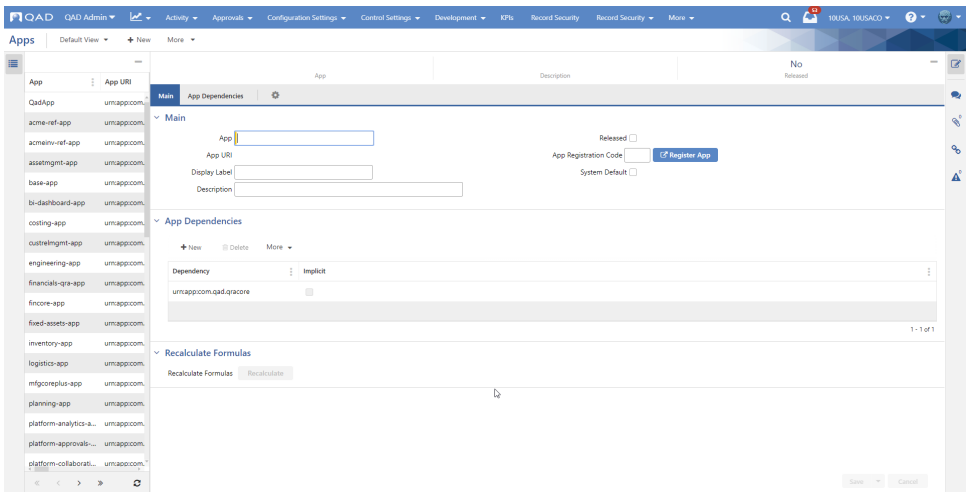
3- Create a custom App

To create a custom app:

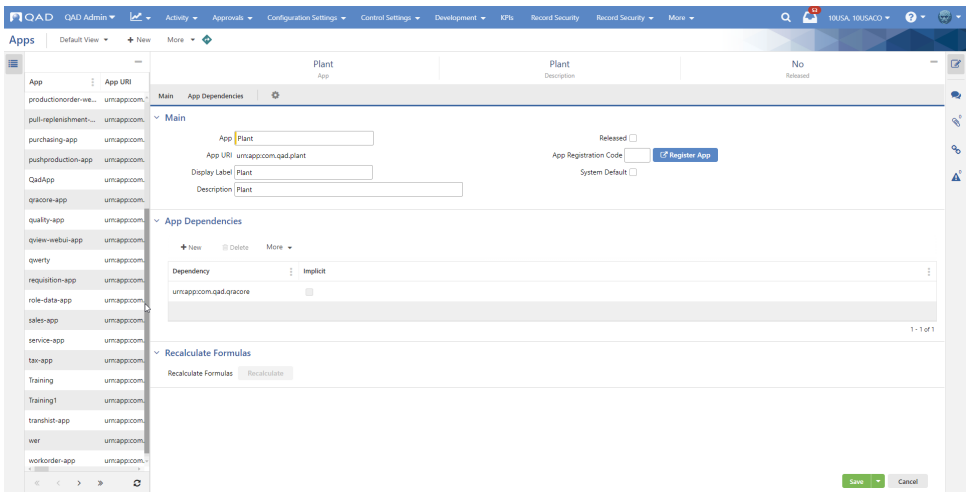
1. Navigate to **Apps** from the menu search.



2. Click the **New** toolbar button.



3. Define the "App" name, "Display Label", "Description", and "System Default".



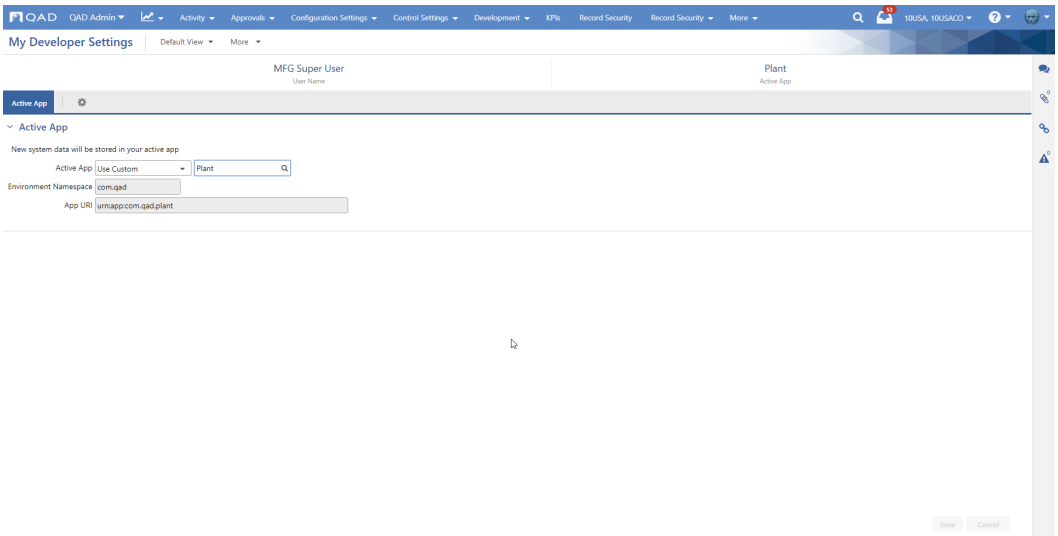
4. Register your app with QAD, and then enter a 4-character code from the QAD App Registration page in the App Registration Code field. Apps that will be shared with others and included in environments with other shared apps must be registered with QAD so that they have a unique app registration code. This unique app registration code is important because it provides the prefix for any of the app's business component physical table names. Note that QAD-provided coded business components do not require registration.
5. Save the 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 **My Developer Settings**.
2. Click the "Active App" drop-down menu, and then select Use Custom.
3. Open the lookup, choose a new App as active, and then click OK.
4. Click Save.

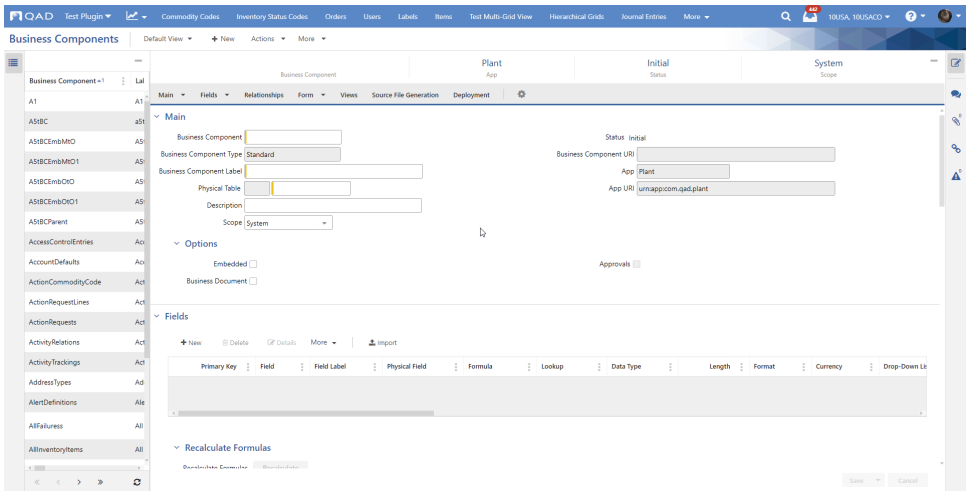


Now, a new App is set as active, and this means that all further secured resources will be created in this App.

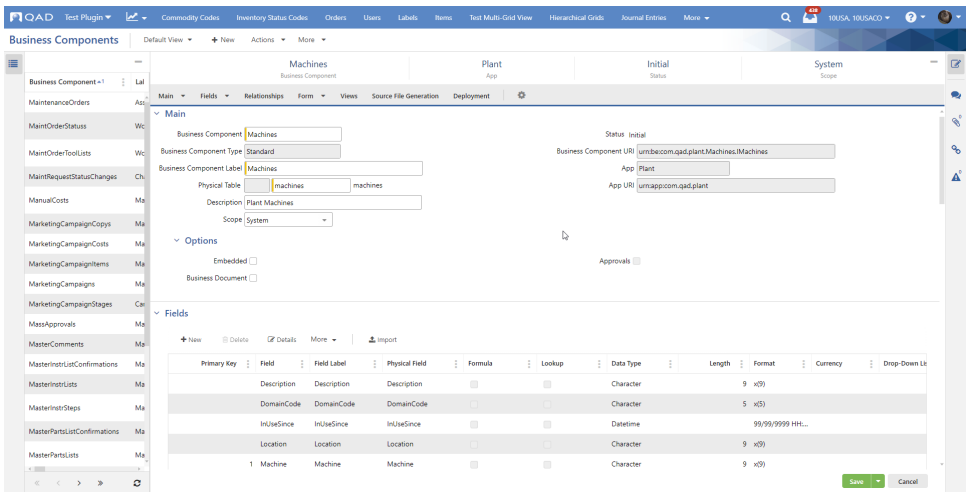
2. Create a business component

1- Create a Business Component

1. Navigate to **Business Components** from the menu search.
2. Click the **New** toolbar button.

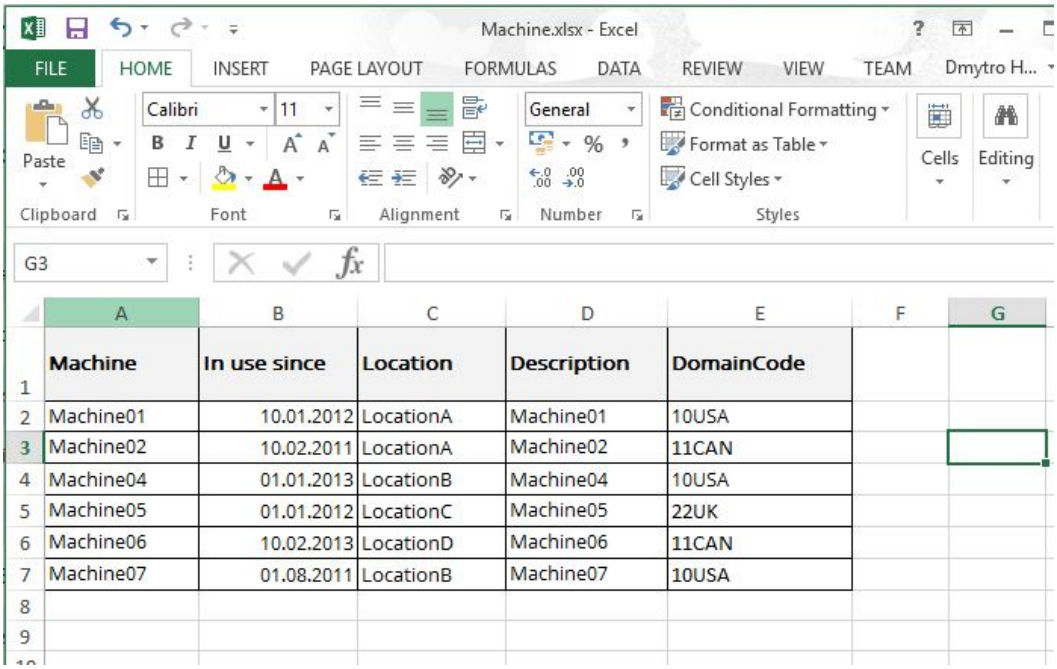


3. Navigate to the **Main** panel.
4. Define the "Business Component" name, "Business Component Label", "Physical Table", "Description", and "Scope". The Business Component URI will be initialized automatically depending on the "Business Component" name.
5. Navigate to the **Fields** panel.
6. Create the fields for this 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", and the third - should be set as "3", and so on).

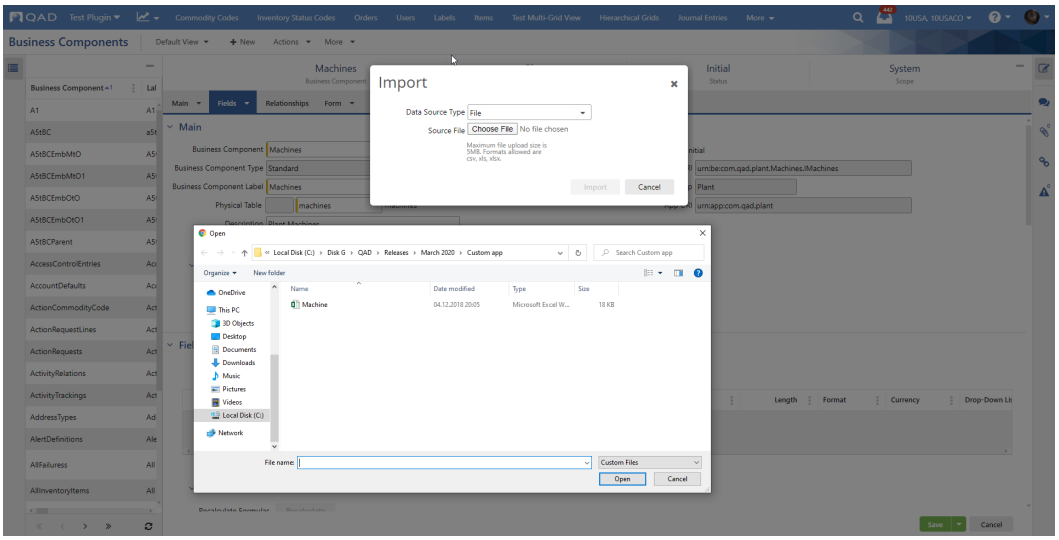


Note: The fields for a Business Component can be created in three ways:

- 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.



	A	B	C	D	E	F	G
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							



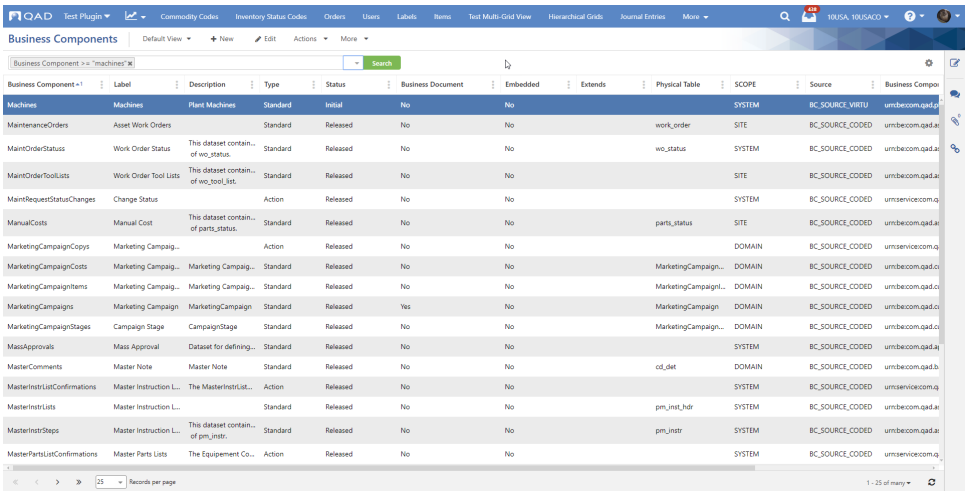
7. Save the Business Component.

Next, you need to create a Form and Hybrid View/Browse for this Business Component.

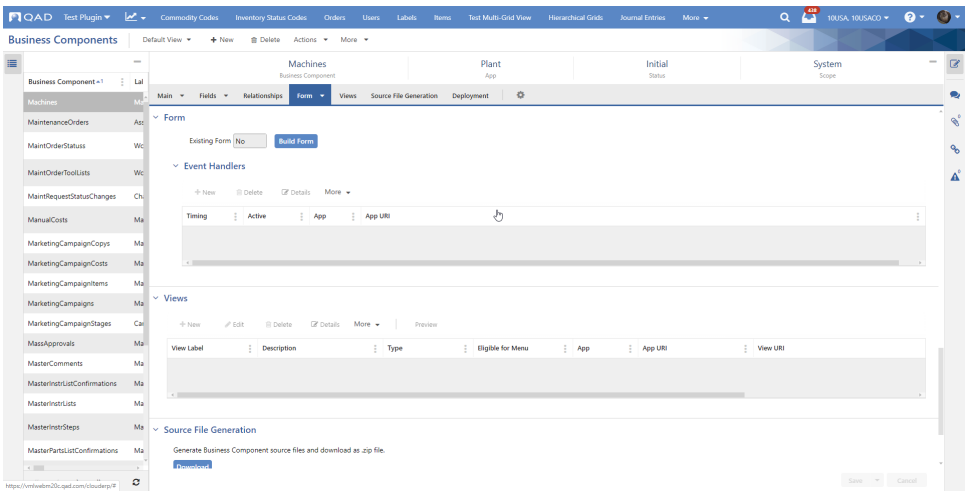
3. Create a view (Form and Hybrid Browse) for a business component

1- Form Builder

1. Navigate to **Business Components** from the menu search.
2. Using the quick search, find the created Business Component.

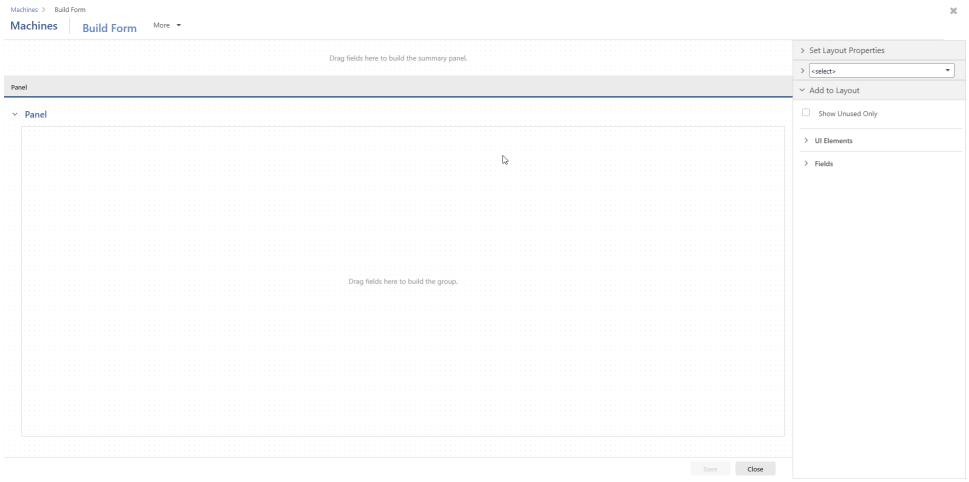


3. Click the **Edit** toolbar button.
4. In the opened screen, you can see the **Form** and **Views** panels that give a possibility to create a Form and Hybrid View for this Business Component.

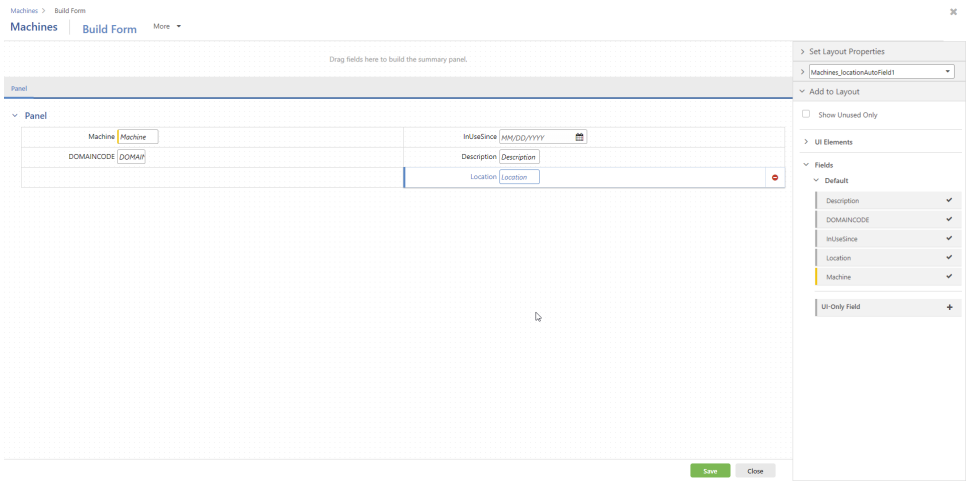


2- Create a Form

1. Navigate to the **Form** panel, and then click the **Build Form** button.



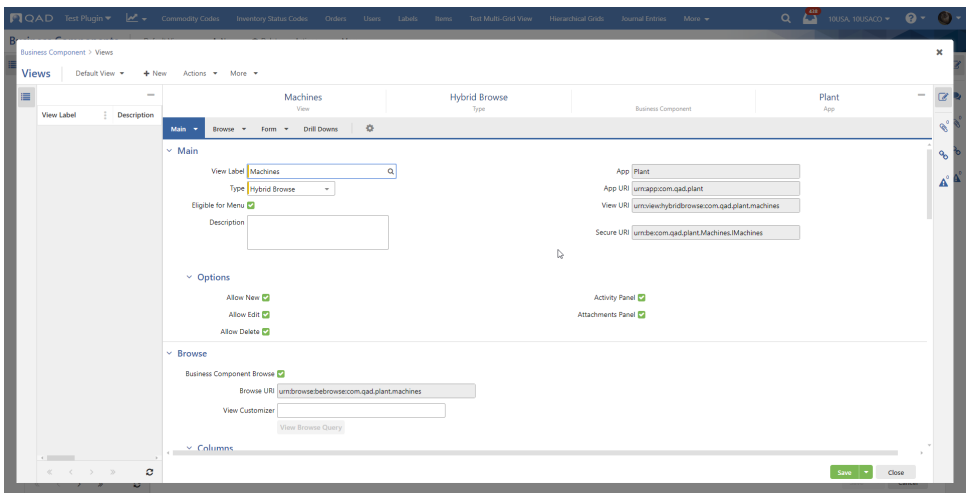
2. Expand the **Fields** sub-section in the **Add to Layout** section, expand the **Default** sub-section, and then drag and drop all fields on the panel.



3. Save the form, and then Close.

3- Create a Hybrid Browse

1. Navigate to the **Views** panel, and then click the **New** grid button.
2. In the opened screen, enter the View Label.



3. Click Save, and then Close.

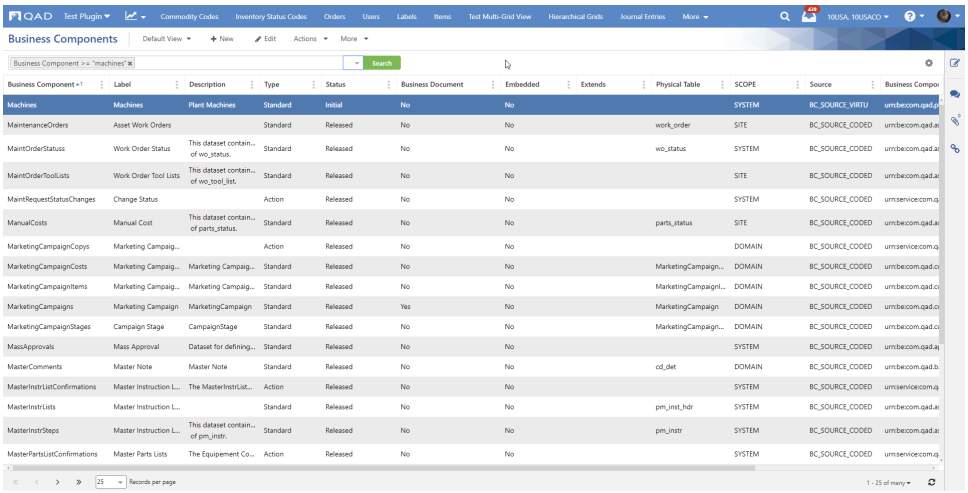
4. Save the Business Component.

The Form and Hybrid Browse are now created for the platform Business Component.

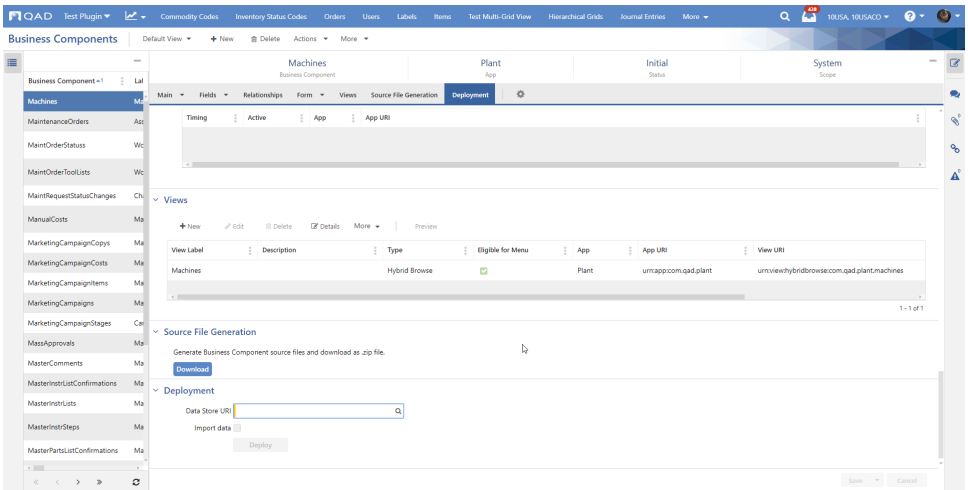
4. Deploy a business component

1- Deploy a platform Business Component

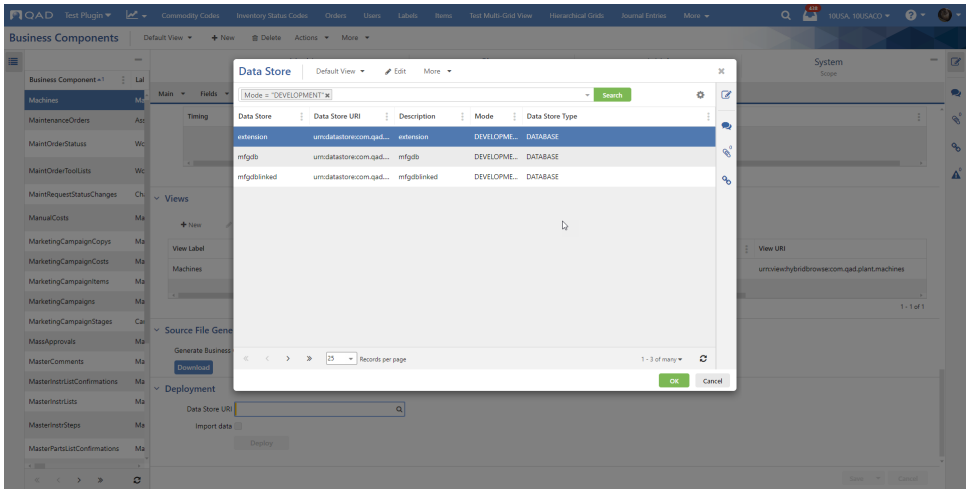
1. Navigate to **Business Components** from the menu search.
2. Using the quick search, find the created Business Component.

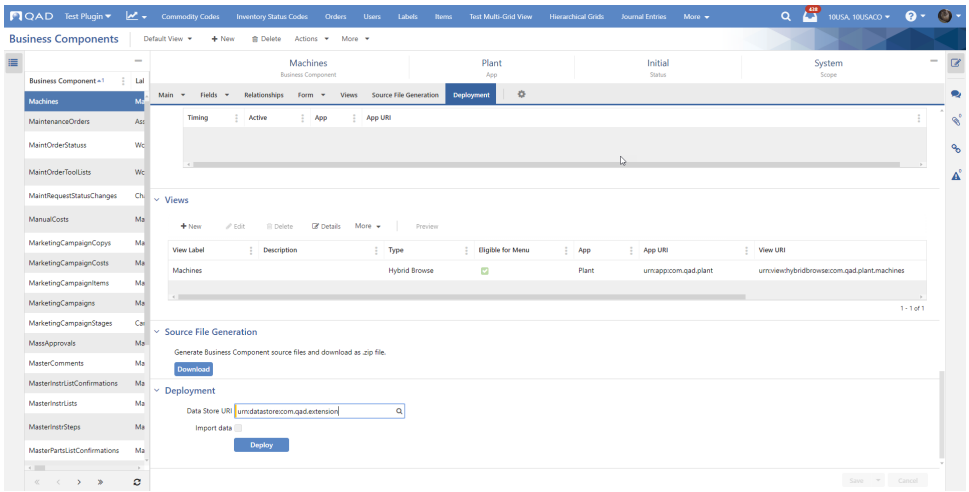


3. Click the **Edit** toolbar button.
4. Navigate to the **Deployment** panel.



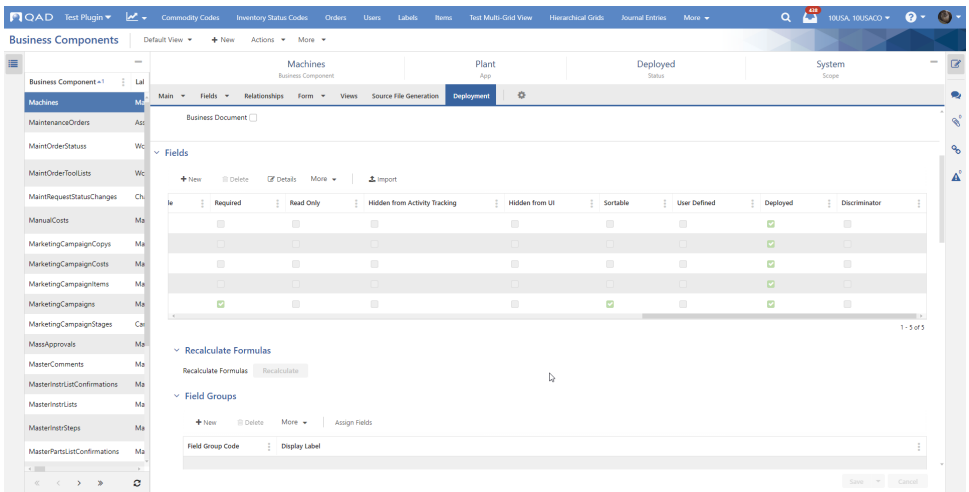
5. Open the lookup for the "Data Store URI" field, choose the data store where the business component should be deployed, and then click OK.





Note: If you use a file for import, select the **Import data** checkbox. The "Filename" field will appear displaying the name of the file that was used for import.

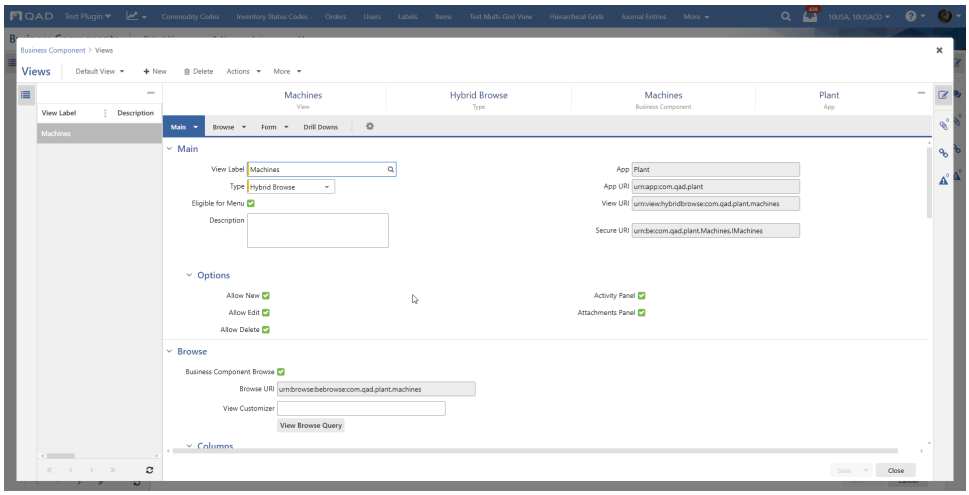
6. Click the **Deploy** button.
   Business Component is successfully deployed, that is represented by the business component status on the Summary Panel, and all fields are marked as "Deployed" now.



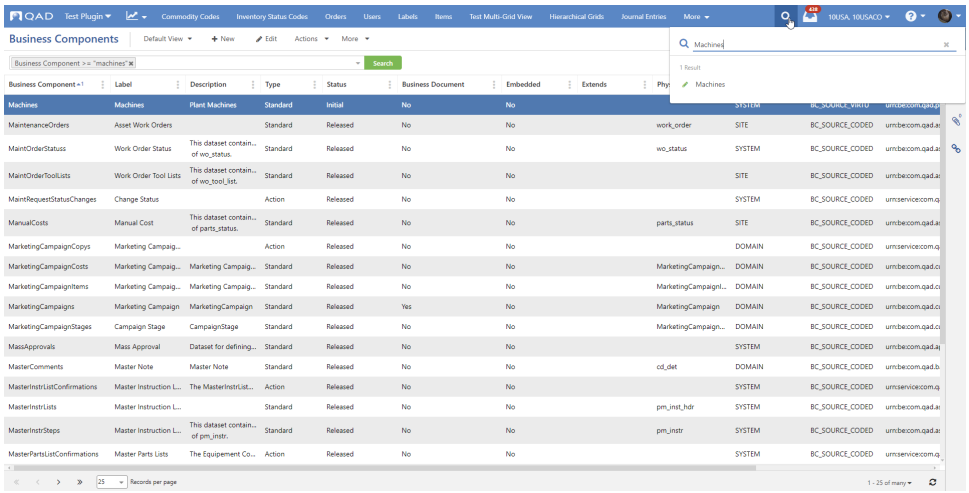
2- Run a view of a Business Component

Once the Business Component is successfully deployed, it is possible to run a view of the Business Component from the menu search by using the View Label that is defined for this Business Component (if the Eligible for Menu checkbox is selected in the Views screen).

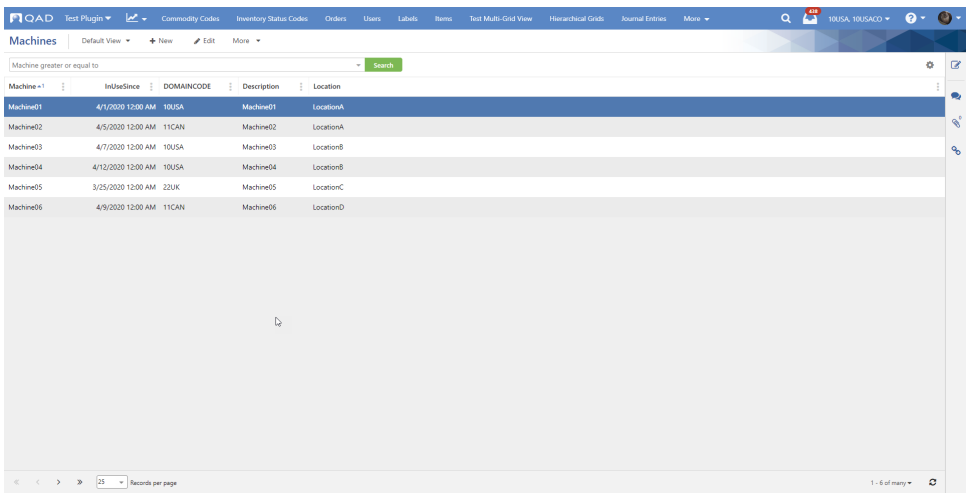
1. Get the label of the view, in this example, it is **Machines**.
2. Be sure the **Eligible for Menu** checkbox is selected.



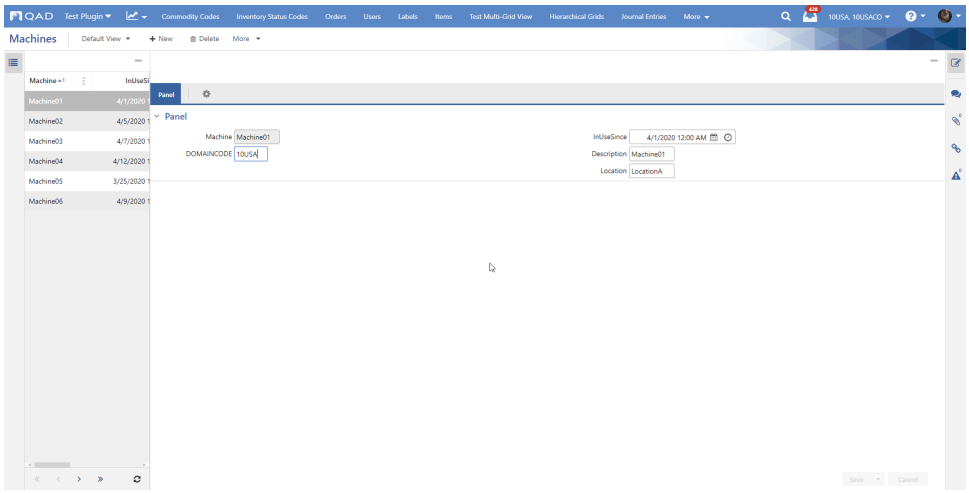
3. Use the menu search to find **Machines**.



4. After running **Machines** from the menu search, the view is opened, including all imported data from the file if any. Or you can add data manually by using the **New** toolbar button.



5. Choose any record and click the **Edit** toolbar button.



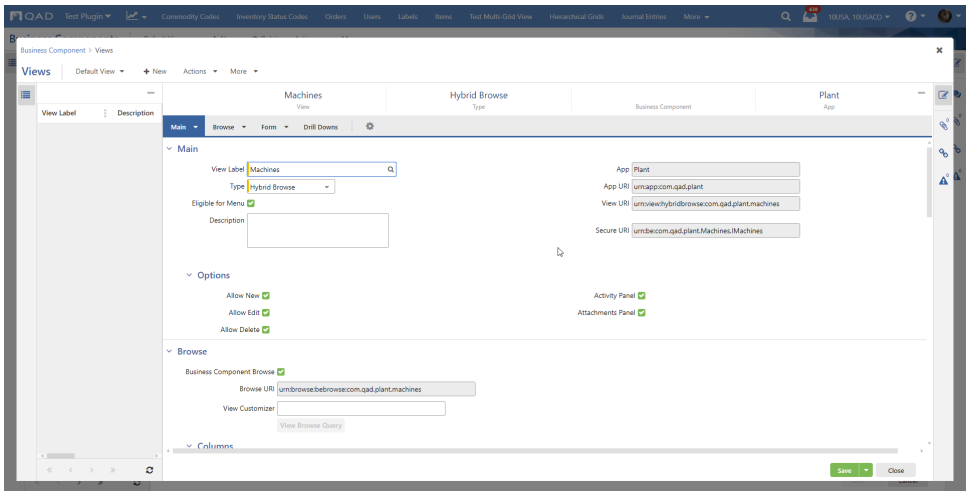
5. Add a business component browse

The Business Component Browse can be created while you build/edit the View.

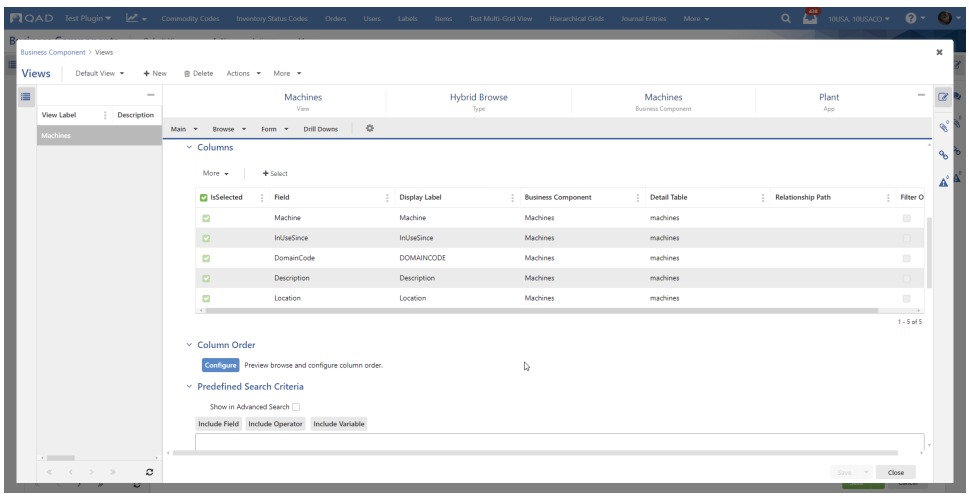
Table of Contents:

1- BC Browse for a single Platform Business Component without relationships

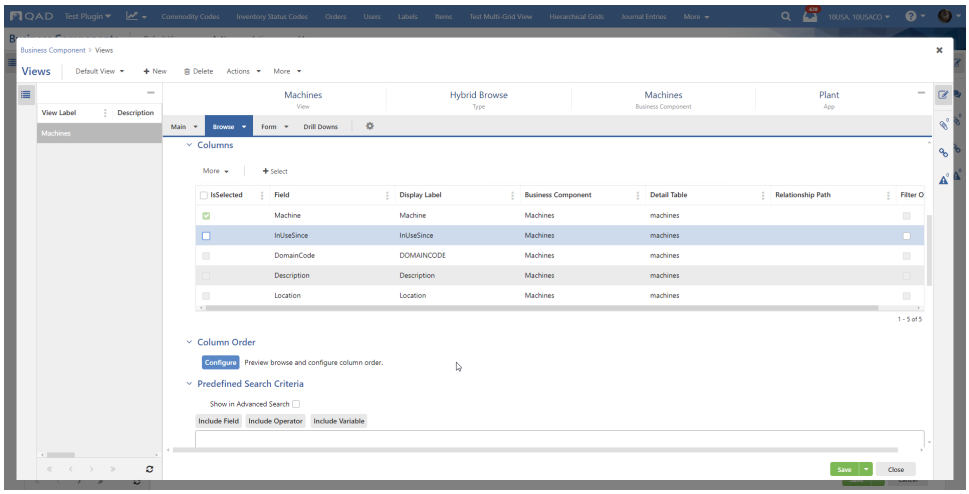
1. Navigate to **Business Components** from the menu search.
2. Using the quick search, find the existing Business Component. In our case, we will use the previously created **Machines** BC.
3. Click the **Edit** toolbar button.
4. Navigate to the **Views** panel, select the previously created **Machines** View, and then click the **Edit** grid button.
5. The **Views** screen opens. Note 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.



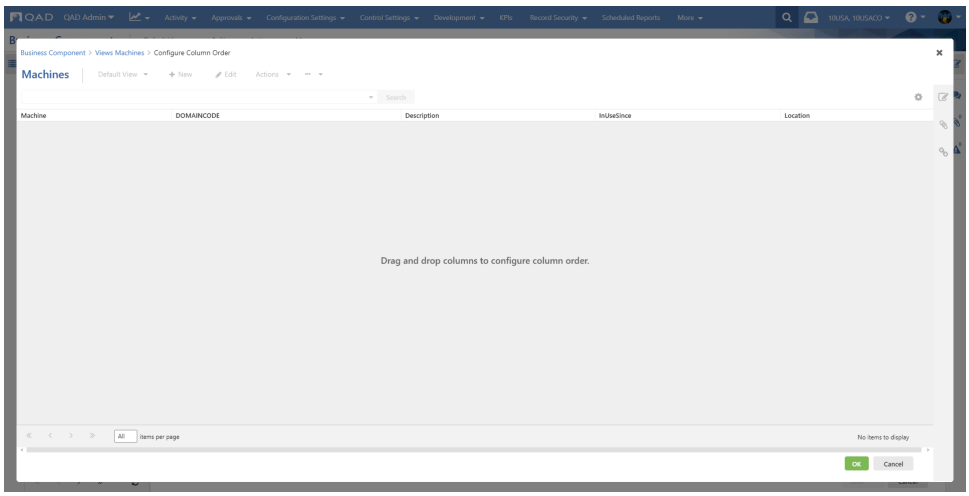
6. Pay attention that the **Columns** grid consists of those fields that were dragged to the Panel in the Form Builder. Fields that were not dragged can be found in the Selector and added to the **Columns** grid. To do this, click the **+Select** button, select the relationship, click **Continue**, and then in the **Select Fields** pop-up window, select the fields, and then click **OK**.



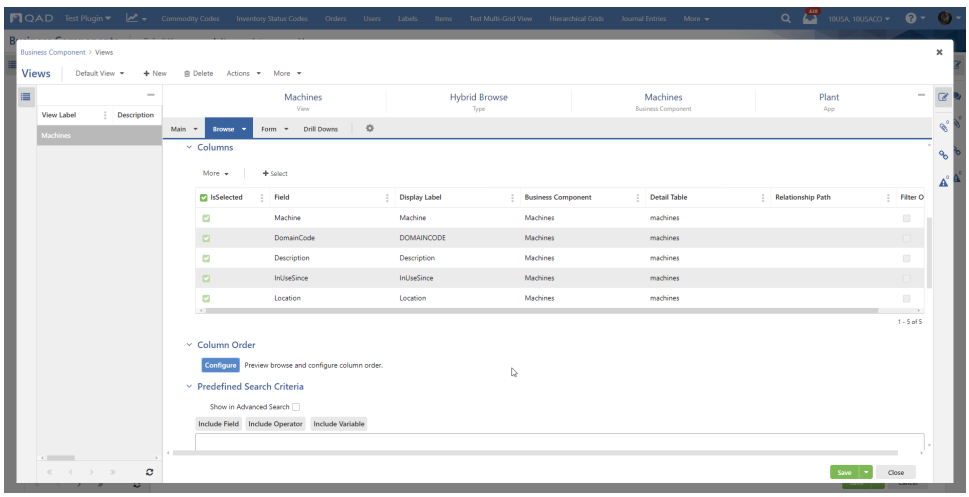
7. The fields can be unselected (except the "Primary Key" field, it is selected and disabled by default).
   Important: The "Primary Key" field is always present in the **Columns** grid even if it was not dragged to the Panel in the Form Builder.



8. Mark the fields you need in the BC Browse as selected.
9. Click the **Configure** button on the **Column Order** panel. The Preview for the Business Component Browse opens.
10. Change the fields' order by dragging-and-dropping them if you want and save the changes.

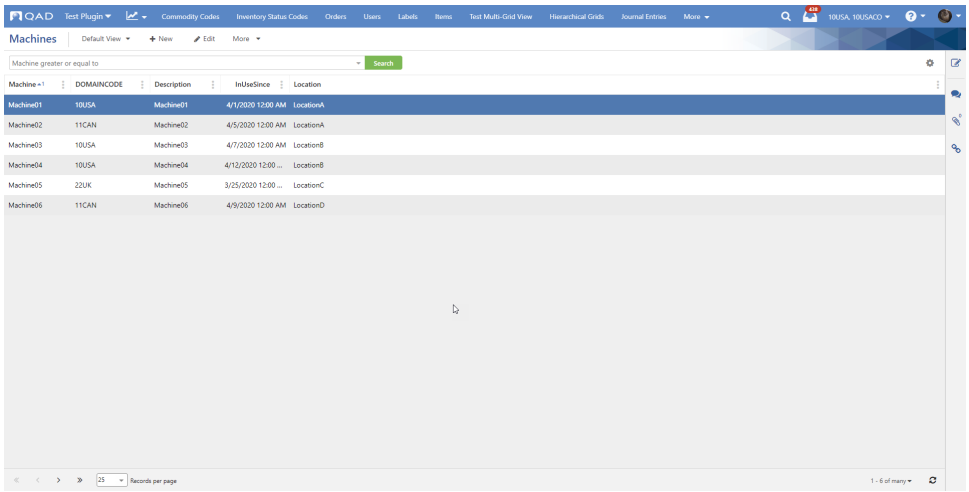


11. See that the fields' order changed in the **Columns** grid.



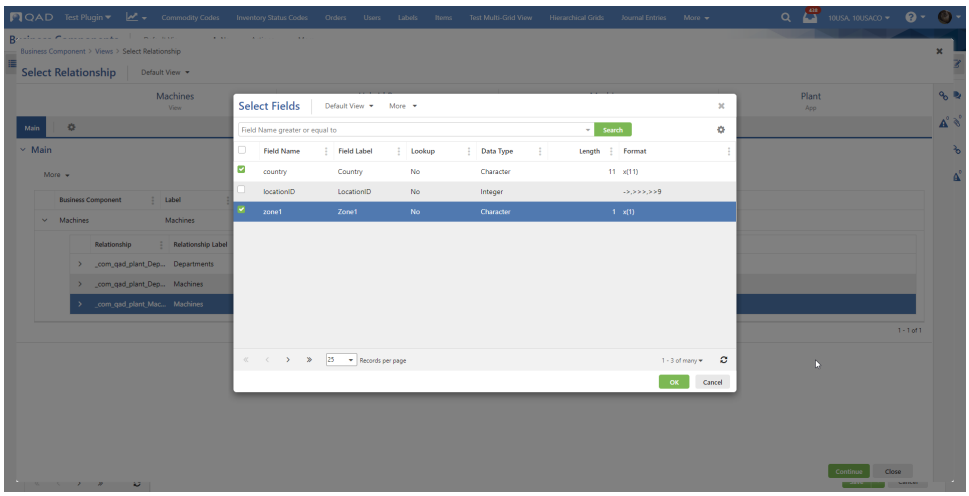
12. Click the **Save** button and close the **Views** screen.

13. The Business Component Browse can be opened from the menu search (only for already deployed Business Components); it is ready for CRUD operations. Note that you may need to refresh the page to see the changed column order.



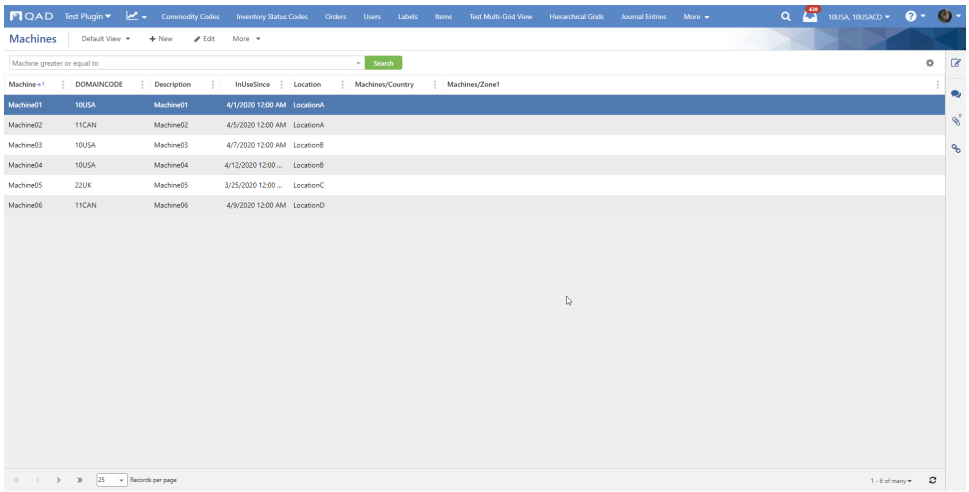
2- BC Browse for Platform Business Component related to another Platform Business Component (One-to-one cardinality)

1. See 6 p., there is a traditional 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, select the relationship, click **Continue**, and then in the **Select Fields** pop-up window, select the fields, and then click **OK**.
4. For a traditional relationship, the "Primary Key" field from the related BC can be selected and added to the BC Browse.



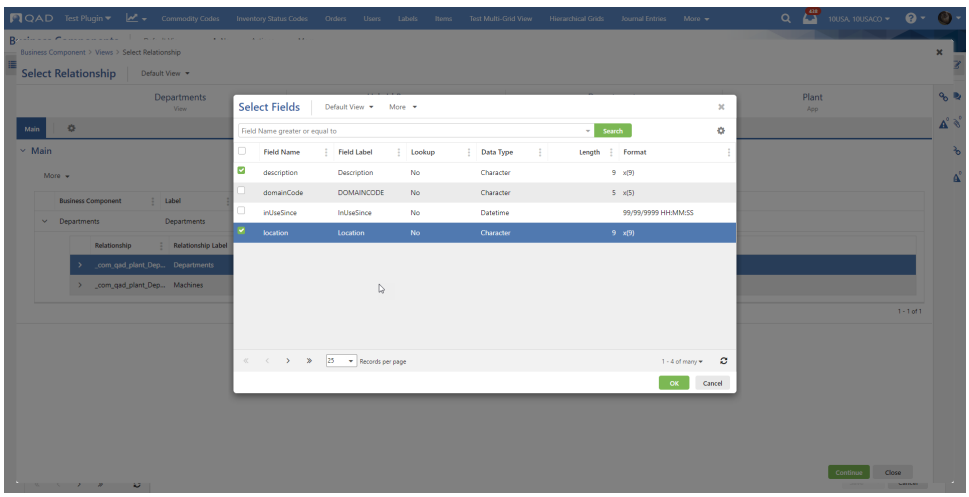
5. Confirm the saving.
6. Only fields that belong to the deployed Business Component will be displayed in the Preview of BC Browse, so you need to deploy both the Main and related BCs if you want to observe all fields you have included.

7. Here, you can see fields from the Main BC and related fields, which are displayed with their appropriate Relationship Name.

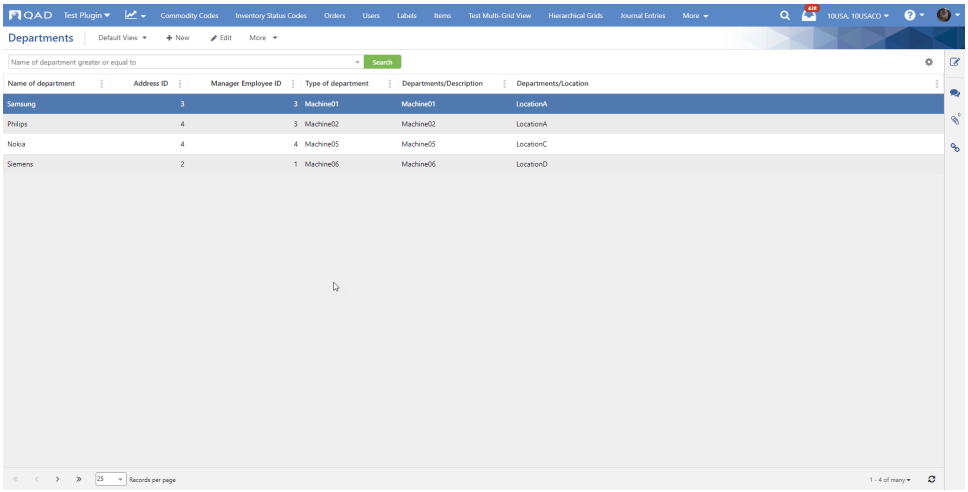


3- BC Browse for Platform Business Component related to another Platform Business Component (Many-to-one cardinality)

1. See 6 p., there is a traditional relationship between the Main BC **Departments** and related BC **Machines**.
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 "Many" 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, select the relationship, click **Continue**, and then in the **Select Fields** pop-up window, select the fields, and then click **OK**.
4. For a traditional relationship, the "Primary Key" field from the related BC can be selected and added to the BC Browse.



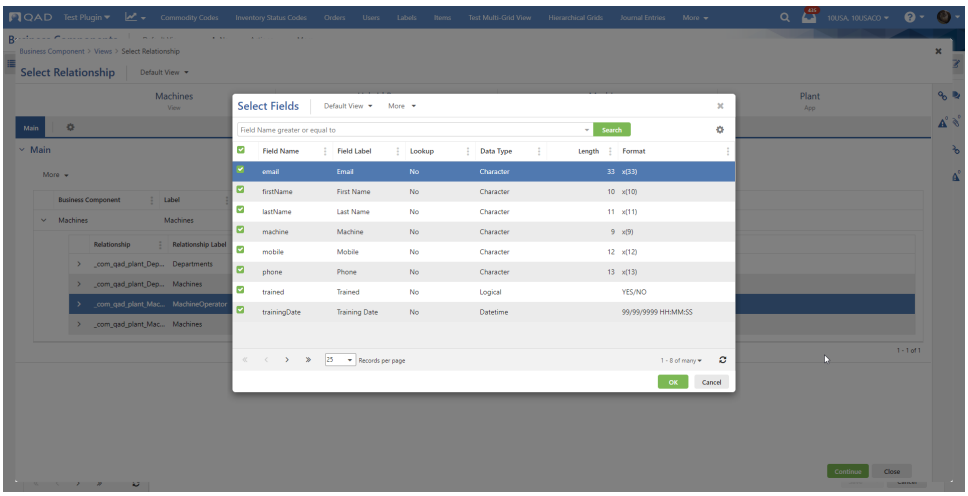
5. Confirm the saving.
6. Keep in mind that only fields that belong to the deployed Business Component will be displayed in the Preview of BC Browse, so you need to deploy both the Main and related BC if you want to observe all fields you have included.
7. Here, you can see fields from the **Departments** BC and related fields, which are displayed with their appropriate Relationship Name.



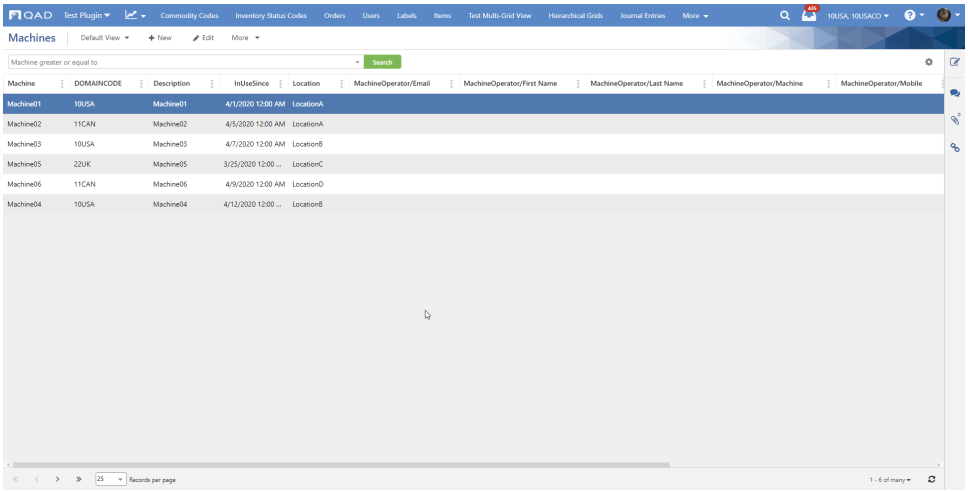
4- BC Browse for Extended Platform Business Component (One-to-one cardinality)

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 a child-parent relationship between the BC **Machines** and embedded BC **MachineOperator**.
2. You can create a Hybrid View with BC Browse for the extended BC **Machines** and all fields from the Main and related Business Components can be included (except the related "Primary Key" field). For this 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, select the relationship, click **Continue**, and then in the **Select Fields** pop-up window, select the fields, and then click **OK**.
4. You can see that the related "Primary Key" field "Employee" is not present and cannot be selected as it was mentioned above.

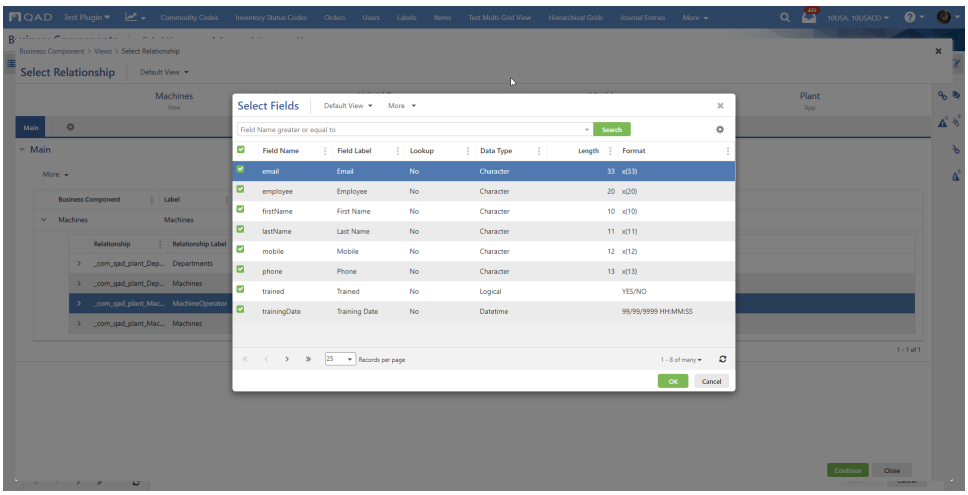


5. Confirm the saving.
6. Only fields that belong to the deployed Business Component will be displayed in the Preview of BC Browse, so you need to deploy both Business Components if you want to observe all the fields you have included.
7. Here, you can see fields from the extended BC **Machines** and its related fields which are displayed with the appropriate Relationship Name.

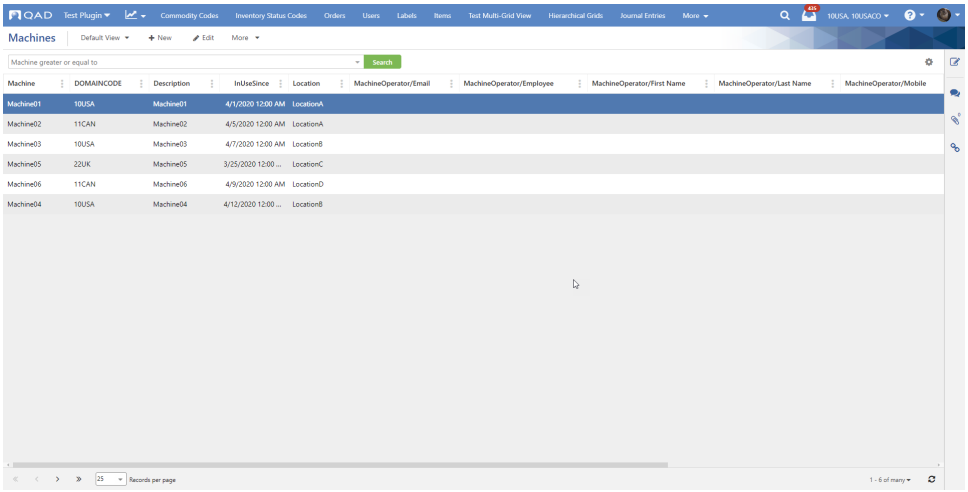


5- BC Browse for Extended Platform Business Component (Many-to-one cardinality)

1. See 7 p., there is a child-parent relationship between the BC **Machines** and embedded BC **MachineOperator** wit h cardinality Many-to-one, where the embedded BC is the "Many" 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.
3. You can create a Hybrid View with BC Browse for the extended BC **Machines** and all fields from the Main and related Business Components can be included (except the related "Foreign Key" field).
4. 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, select the relationship, click **Continue**, and then in the **Select Fields** pop-up window, select the fields, and then click **OK**.
5. You can see that the related "Foreign Key" field is not present and cannot be selected as it was mentioned above.



6. Confirm the saving.
7. 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.
8. Here, you can see fields from the extended BC **Machines** and its related fields which are displayed with the appropriate Relationship Name.


QAD
Test Plugin
Commodity Codes
Inventory Status Codes
Orders
Users
Labels
Items
Test Multi-Grid View
Hierarchical Grids
Journal Entries
More
10USA, 10USACO
Machines
Default View
New
Edit
More
Machine greater or equal to
Search
Machine
DOMAINCODE
Description
InUseSince
Location
MachineOperator/Email
MachineOperator/Employee
MachineOperator/First Name
MachineOperator/Last Name
MachineOperator/Mobile
Machine01 10USA Machine01 4/1/2020 12:00 AM LocationA
Machine02 11CAN Machine02 4/5/2020 12:00 AM LocationA
Machine03 10USA Machine03 4/7/2020 12:00 AM LocationB
Machine05 22UK Machine05 3/25/2020 12:00 ... LocationC
Machine06 11CAN Machine06 4/9/2020 12:00 AM LocationD
Machine04 10USA Machine04 4/12/2020 12:00 ... LocationB
25
Records per page
1 - 6 of many

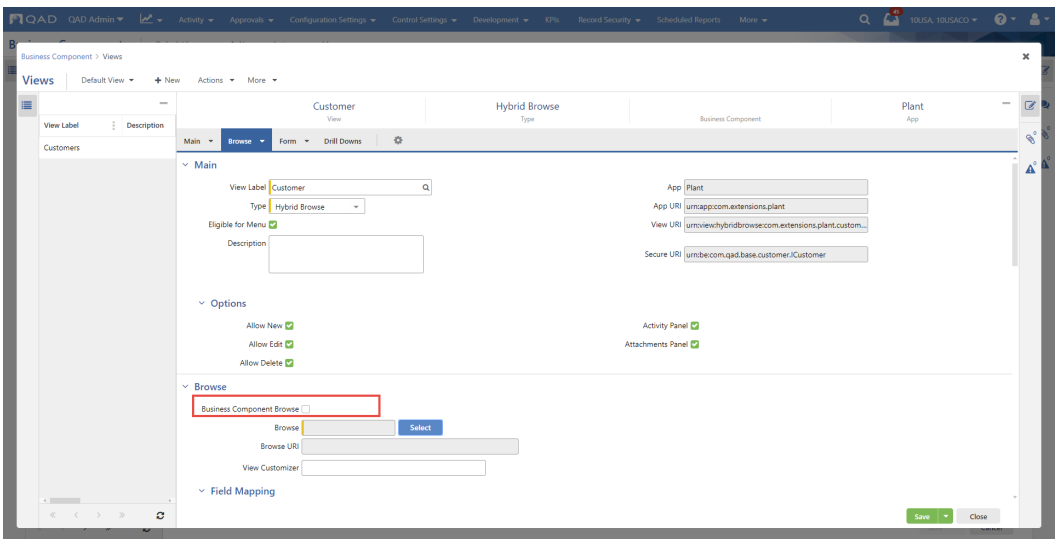
5.1. Create a Standard Browse

Introduction

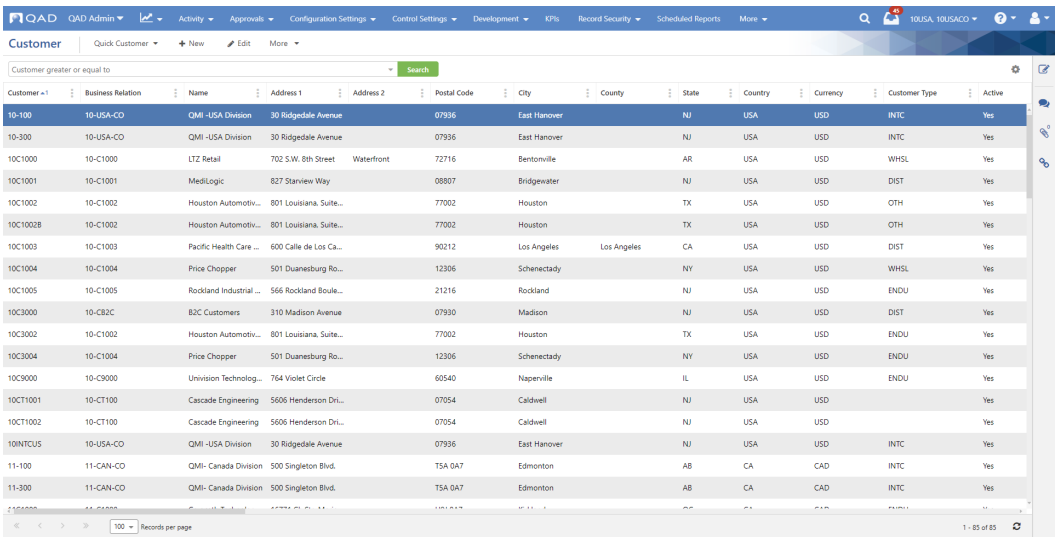
Standard Browse is a browse that shows all fields of Business Component.

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 clear.

2. **View Label**. Displays a label for the View.
3. **Browse**. Click the **Select** button to select the browse resource.

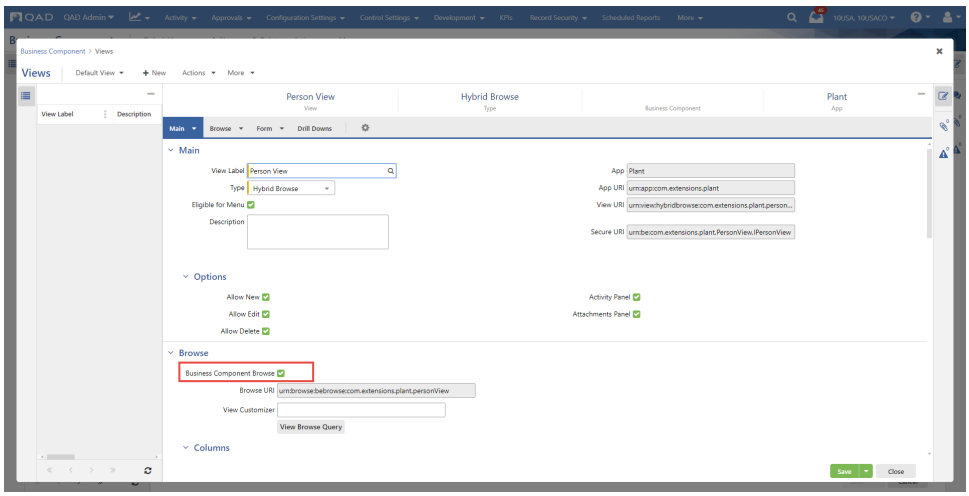
5.2. Create a Custom Browse

Introduction

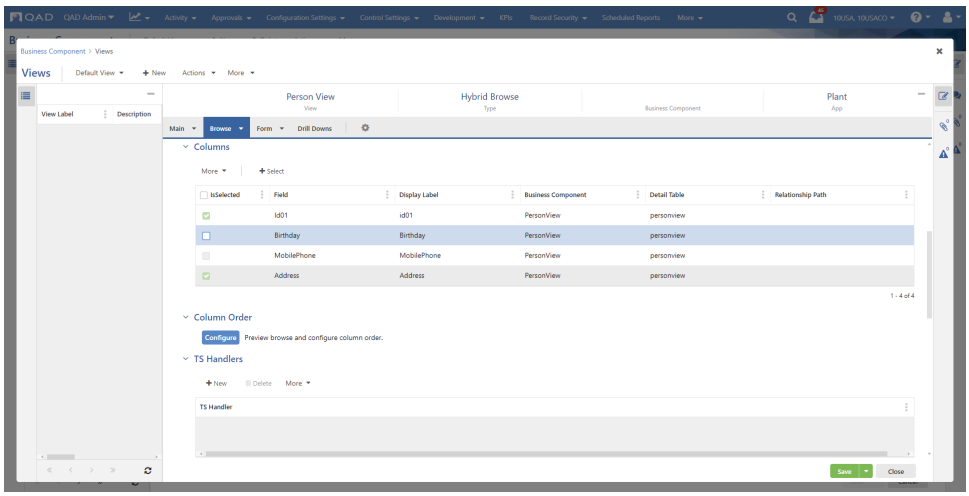
Custom Browse is a definition of a browse and contains information about which fields from the BC and relationships can be shown on the browse page.

Screen Shots

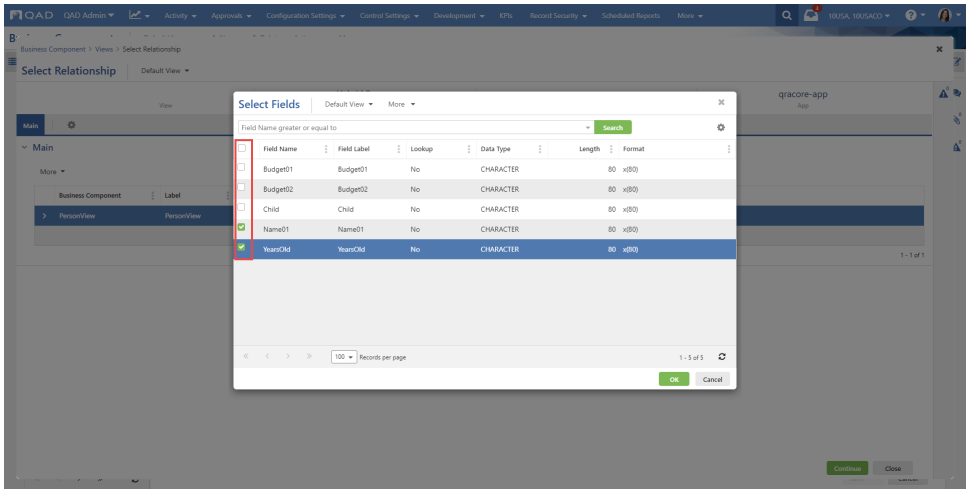
1. In the Form Builder, you can define a View with the Custom Browse support. For a newly created View, the list of fields is taken from the Business Component Form.



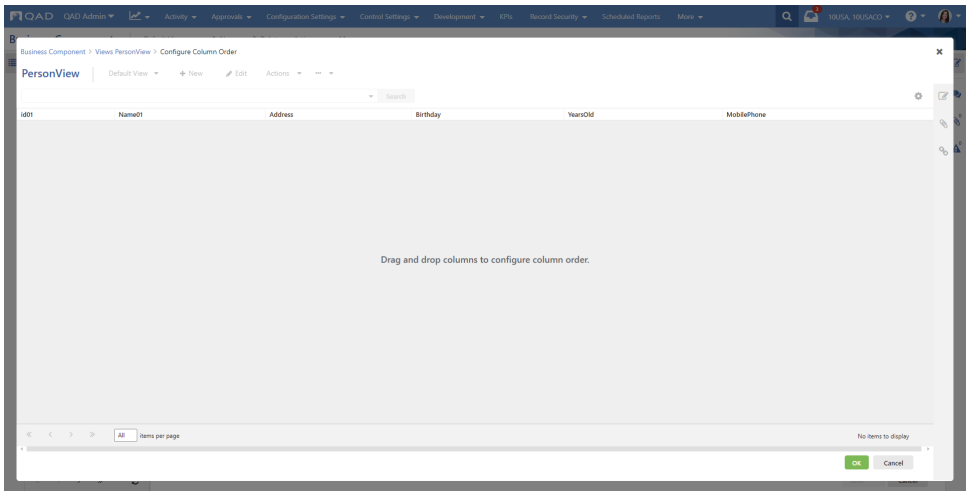
2. To define or modify the list of fields, select the fields in the grid.



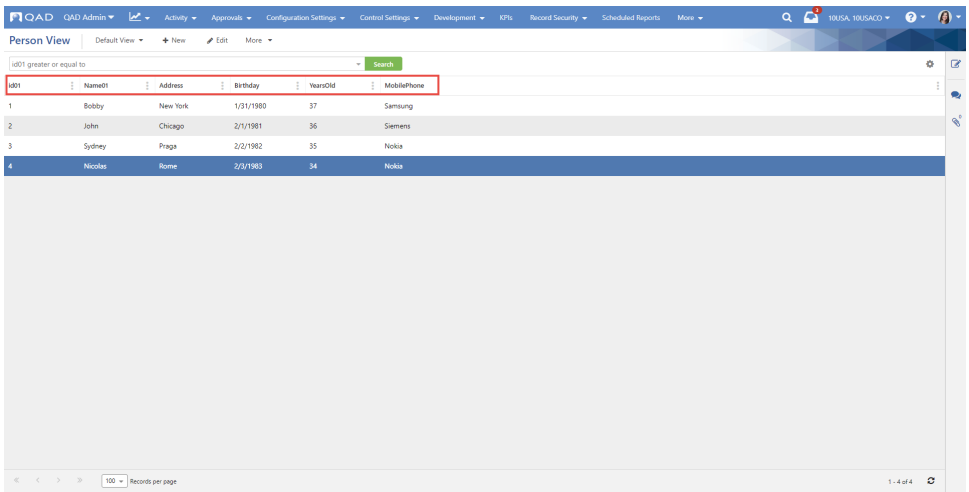
3. To add new fields, click the **+Select** button, select the relationship, click **Continue**, and then in the **Select Fields** pop-up window, select the fields, and then click **OK**.



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 a label for the View.
3. **+Select**. Click the button to add not selected fields to the browse.
4. **Configure**. Click the button to reorder the fields of a browse.

6. Create a relationship between two business components

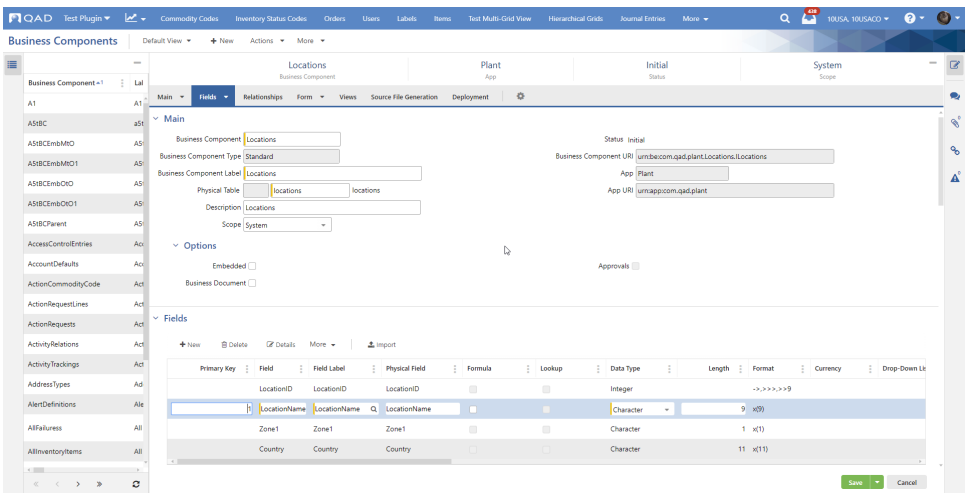
There are two variants of cardinality for traditional (non-extension) Relationships between Business Components:

1- Cardinality: One-to-one

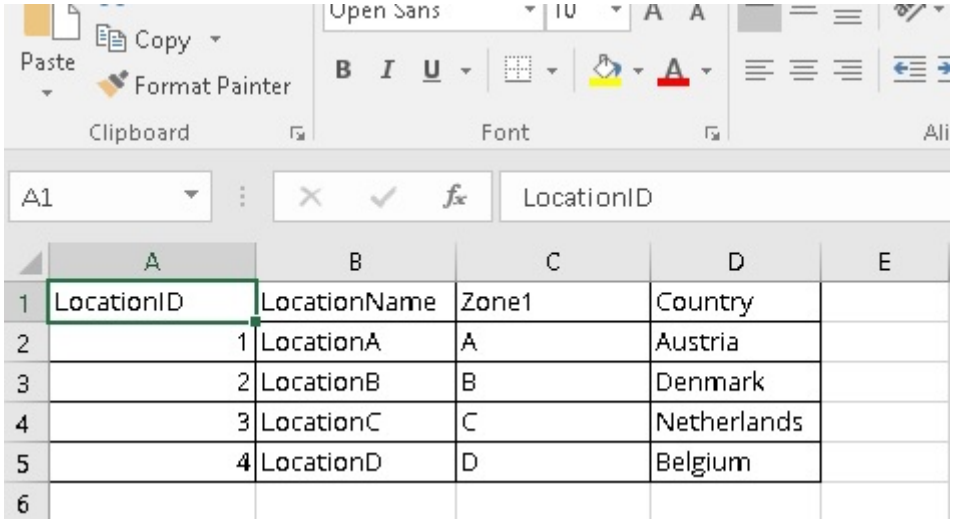
Let's create a Business Component named **Locations** and relate it to the **Machines** BC that was created in our previous examples.
One Machine is located in one appropriate Location. So it is the One-to-one cardinality.

Step 1 - Create the platform BC, which should be related to the Main platform 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 the 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.



This example uses import from the previously prepared Excel file:

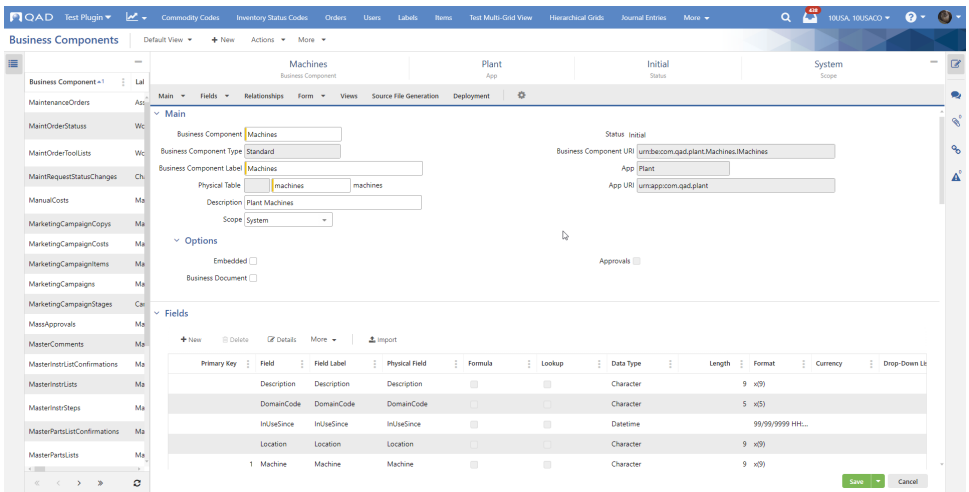


	A	B	C	D	E
1	LocationID	LocationName	Zone1	Country	
2	1	LocationA	A	Austria	
3	2	LocationB	B	Denmark	
4	3	LocationC	C	Netherlands	
5	4	LocationD	D	Belgium	
6					

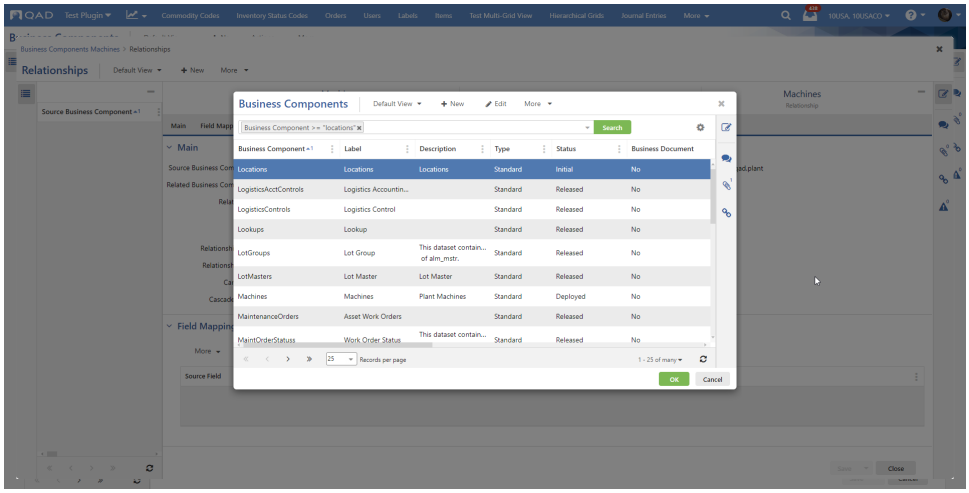
6. If you want to create a Relationship with One-to-one cardinality, the Primary Key field in the related BC should match the Primary Key field in the Main BC **by quantity/order and datatype.** In other case, the cardinality Many-to-one will be set.
7. Save the Business Component.

Step 2 - Create a Relationship between the Main platform BC Machines and platform BC Locations with One-to-one cardinality

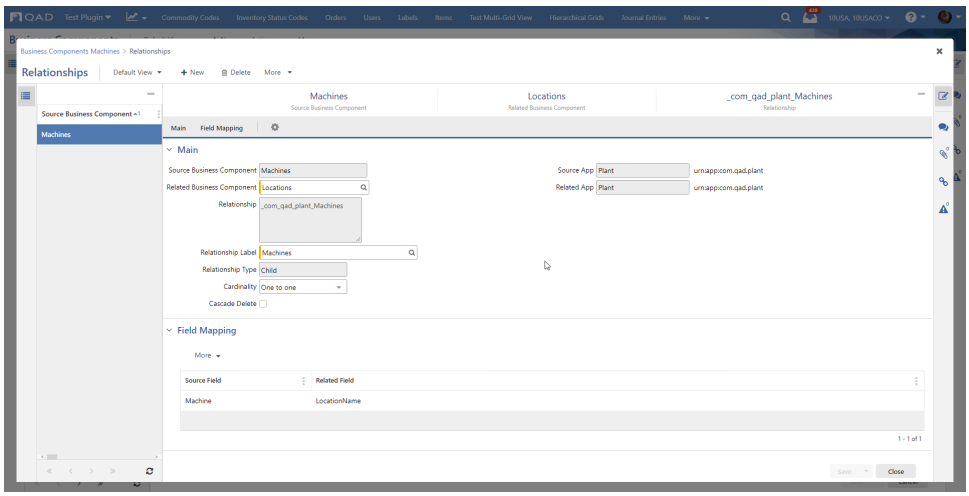
1. Find the Main platform BC **Machines** using the quick search.
2. Click the **Edit** toolbar button. Maintenance View for the **Machines** BC will open.



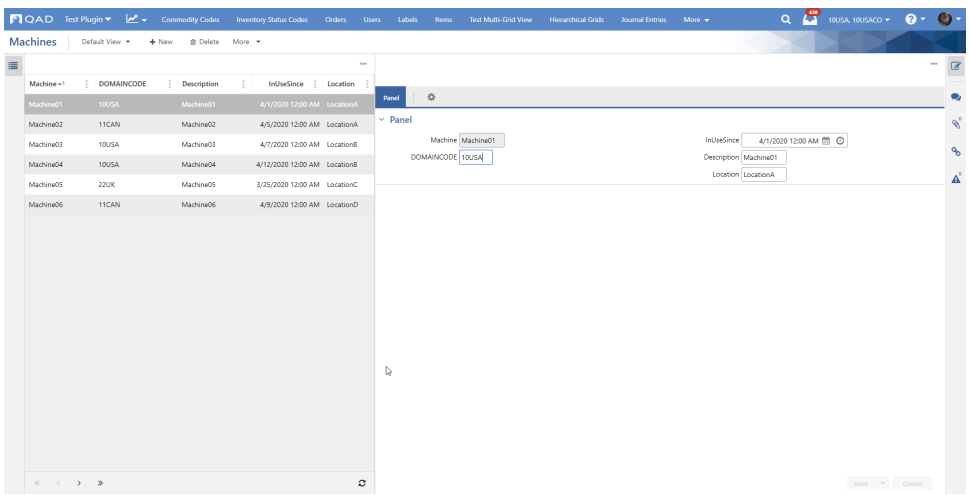
3. Open the **Relationships** panel in the **Business Components** screen, and then click the **New** button.
4. Click the lookup in the "Related Business Component" field and, in the browse, set the criteria for the platform BC that is aimed to be related with the Main platform BC **Machines**. In our 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, and then click **Close**.



8. Save the business component.
9. After that, build the Form and Hybrid View for the related BC **Locations** before its deployment.
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 deployment 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.

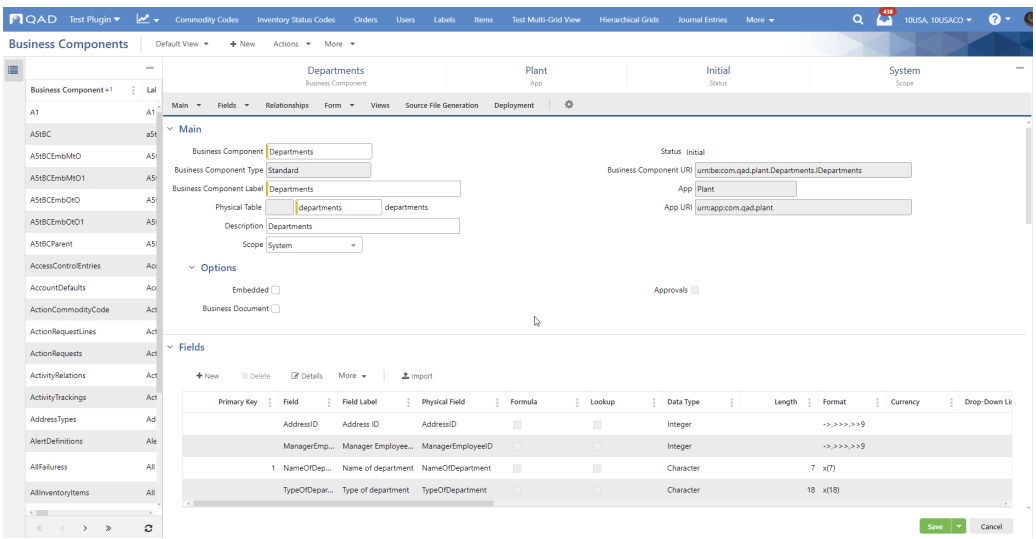


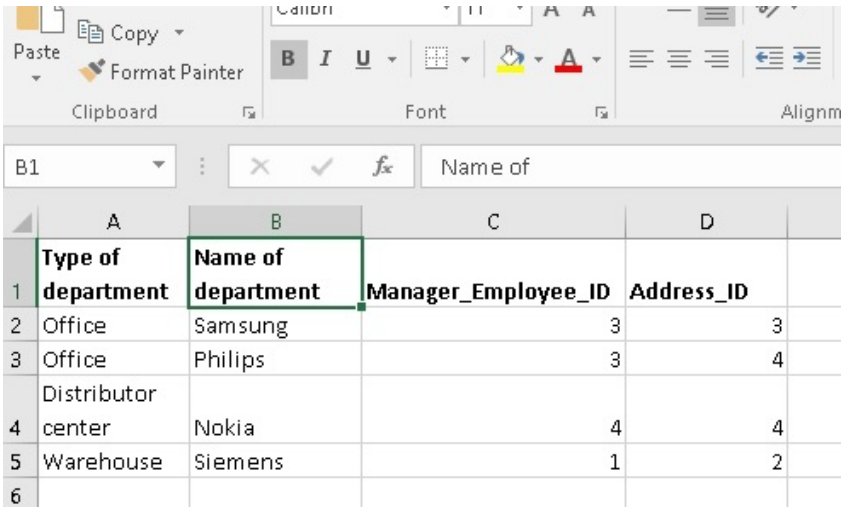
2- Cardinality: Many-to-one

Let's look at the situation when Many-to-one cardinality is used: some types of Departments are served by one specific Machine.
At start, we need to create the **Departments** Business Component and repeat the steps mentioned above.

Step 1 - Create the platform BC, which should be the Main platform 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" field 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 the previously prepared Excel file.
7. Save the Business Component.

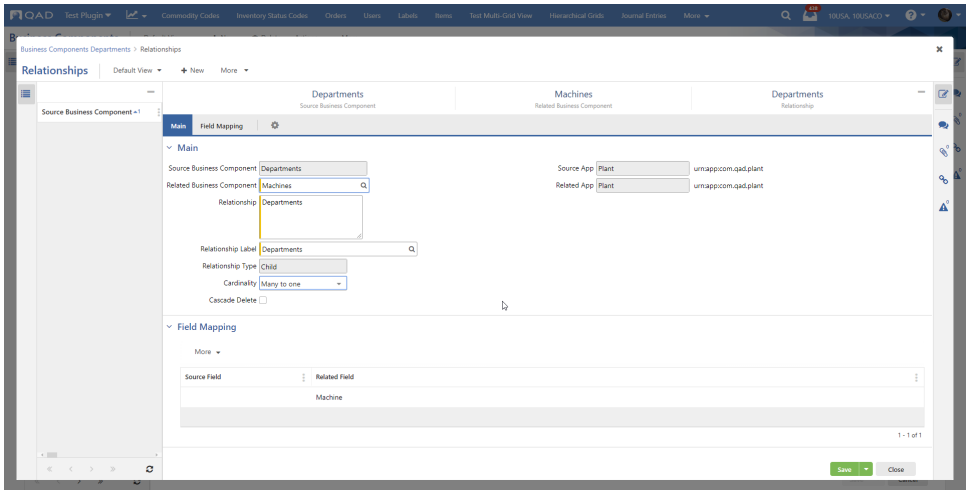




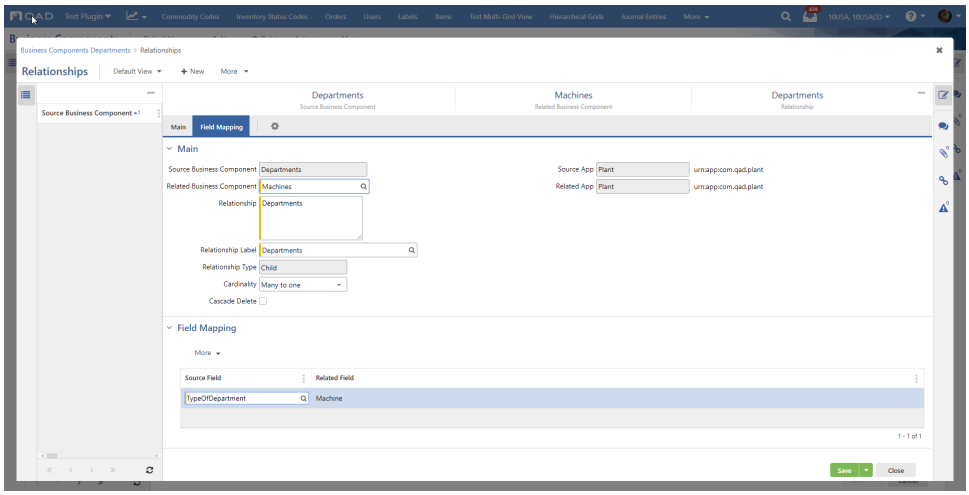
Step 2 - Create a Relationship between the Main platform BC Departments and platform BC Machines with Many-to-one cardinality

1. Find the Main platform BC **Departments** using the quick search.
2. Click the **Edit** toolbar button. Maintenance View for the BC **Departments** will open.
3. Open the **Relationships** panel in the **Business Components** screen, and then click the **New** button.

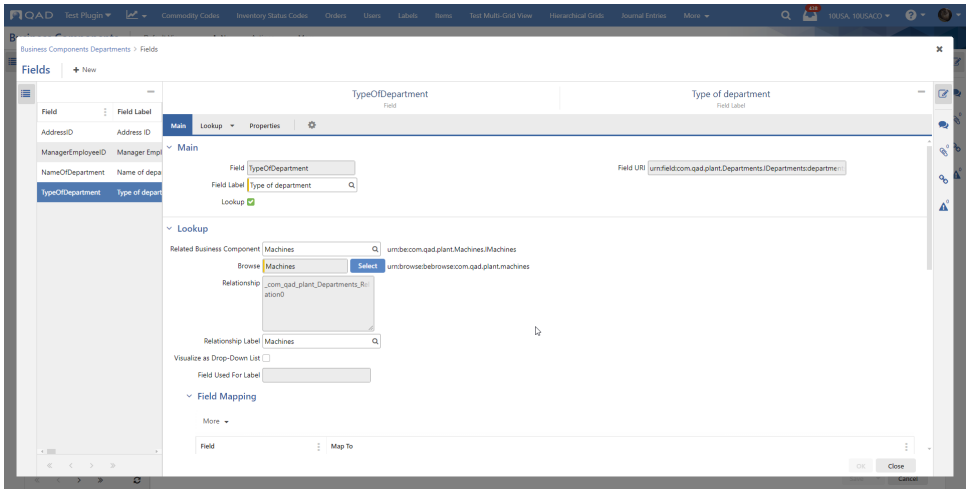
4. Click the lookup in the "Related Business Component" field and, in the browse, set the criteria for the platform BC that is aimed to be related with the Main platform BC **Departments**. In our case, it is **Machines** BC. Then, click the **Search** button.
5. Confirm the selected BC by clicking **OK**.
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 a Relationship with cardinality Many-to-one, change it by selecting it from the "Cardinality" drop-down list.



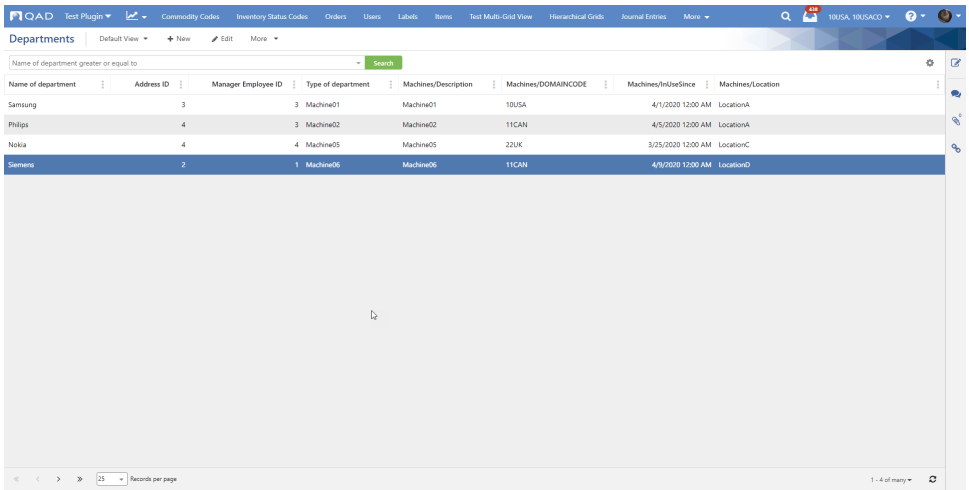
7. Click the lookup in the Source Field, select the "Type of department" field as Foreign Key in this Relationship, and then click **OK**.

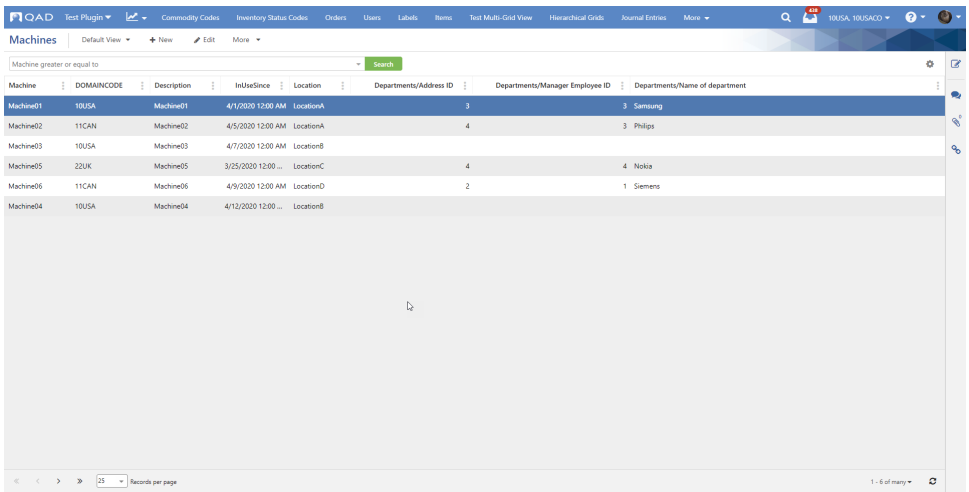


8. Save the Relationship, and then click **Close**.
9. Save the business component.
10. After that, build the Form and Hybrid View for the Main BC **Departments** before its deployment.
11. You can create Hybrid View with BC Browse **for the Departments BC:** all related fields can be included (case with Many-to-one cardinality, where "Many" side is BC with BC Browse).
12. You can set a lookup for a specific field on the **Fields** panel by clicking the **Details** button for the field. Then, select the **Lookup** checkbox and choose the Related Business Component from the lookup.

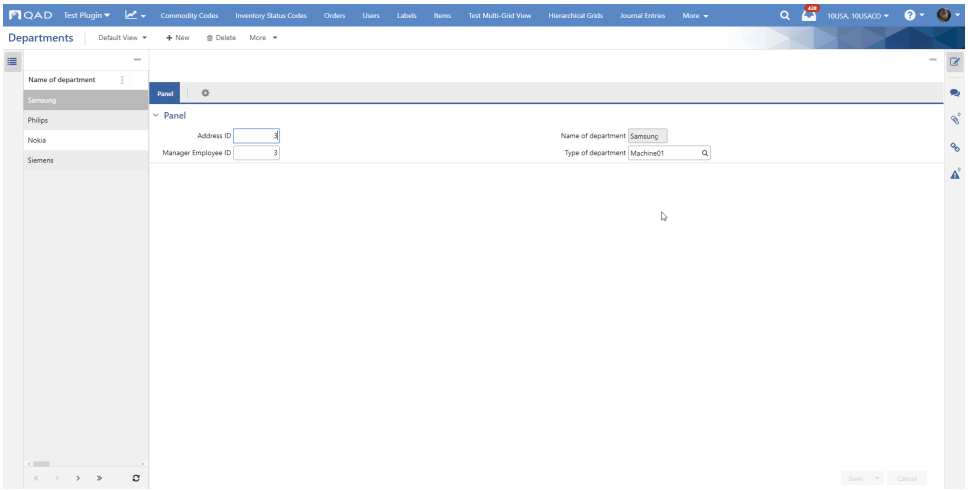


13. Deploy the Main BC **Departments**.
14. You can create a Hybrid View with BC Browse **for the related Machines BC**: all related fields can be included (case with Many-to-one cardinality, where "One" side is BC with BC Browse).
15. Open the Preview for the Main or related BC. Their BC Browses display included fields if they were set into BC Browse.





16. Open the 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 **Machines** BC panel.
17. The 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.

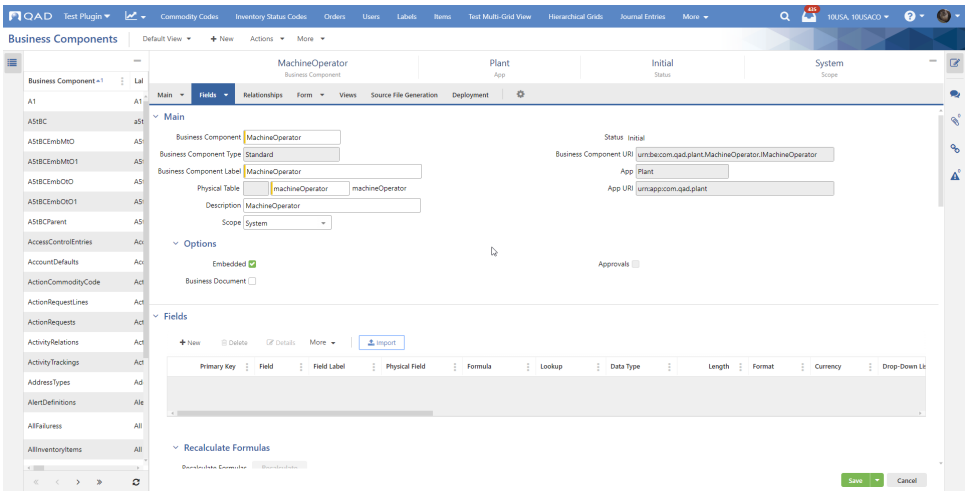


18. After that, the Main and related BCs are ready to be opened and filled with records accordingly.

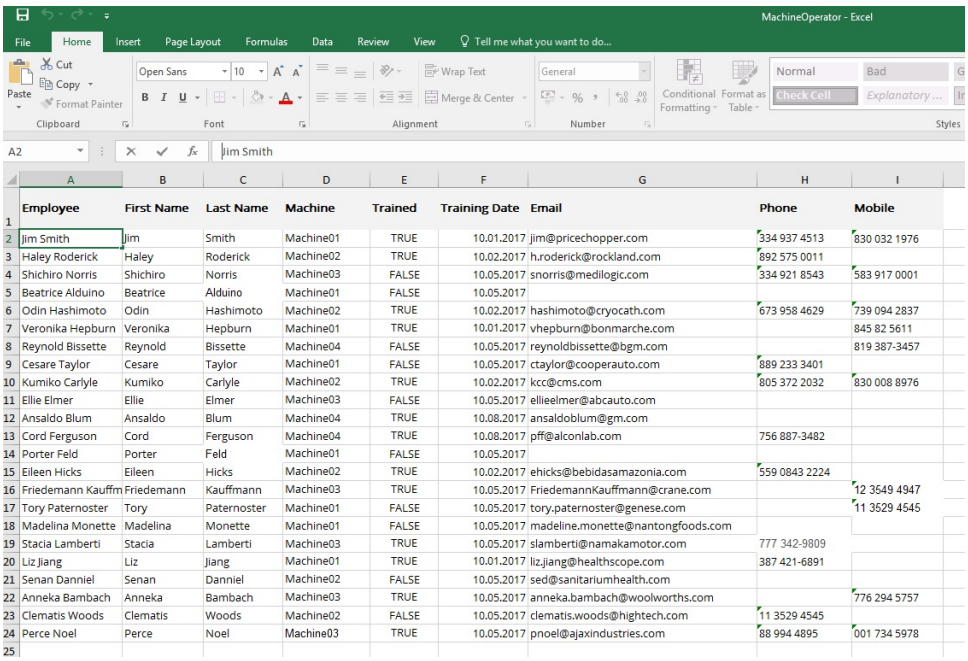
7. Extend a business component with an embedded BC

Step 1 - Create the platform embedded BC which is aimed to extend the Main Platform BC

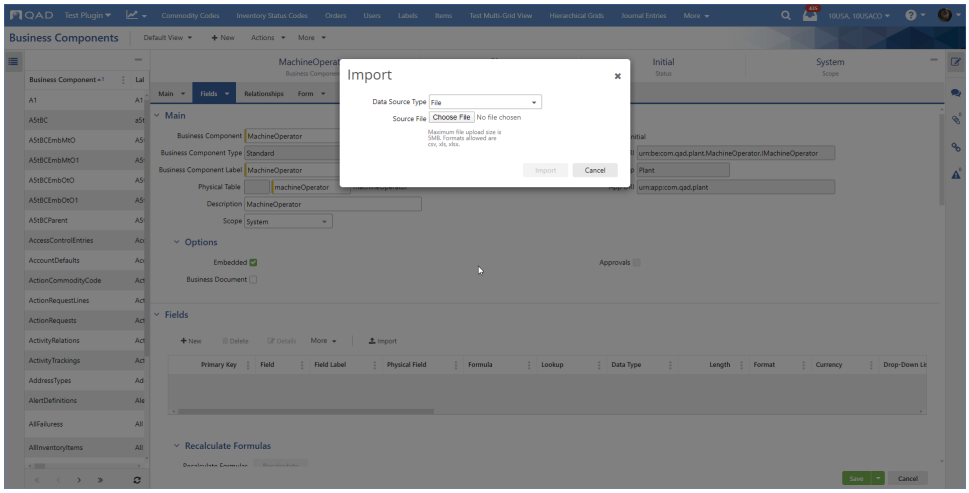
1. Navigate to **Business Components** from the menu search.
2. Click the **New** toolbar button. The "App URI" is already initialized and equals the active App in the system.
3. Navigate to the **Main** panel and define the "Business Component" name, "Business Component Label", "Physical Table", "Description", "Scope", and select the **Embedded** checkbox. The "Business Component URI" will be initialized automatically depending on the "Business Component" name.



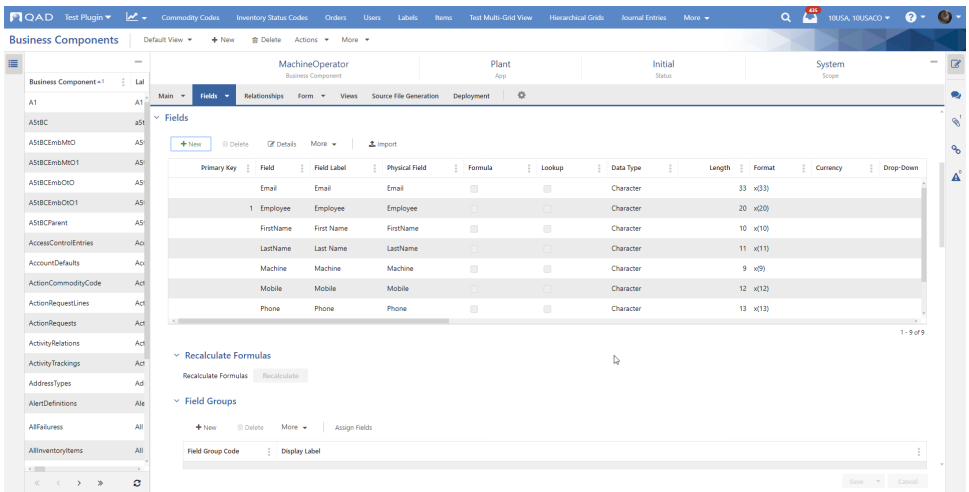
4. Navigate to the **Fields** panel and create the fields for the Business Component. At least one field should be marked as "Primary Key" (set value "1" to "Primary Key").
5. In 2 p., a few ways for Fields creation were mentioned. The example below uses import from the previously prepared Excel file:



	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 child-parent relationships:
   - One-to-one or Many-to-one
   - If you want to create a child-parent relationship with cardinality One-to-one, 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 Many-to-one 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.

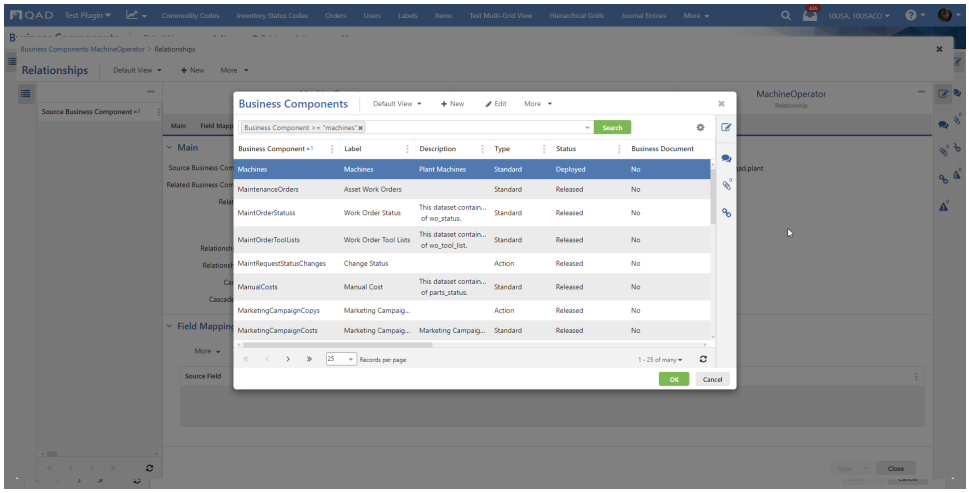


7. Save the Business Component.

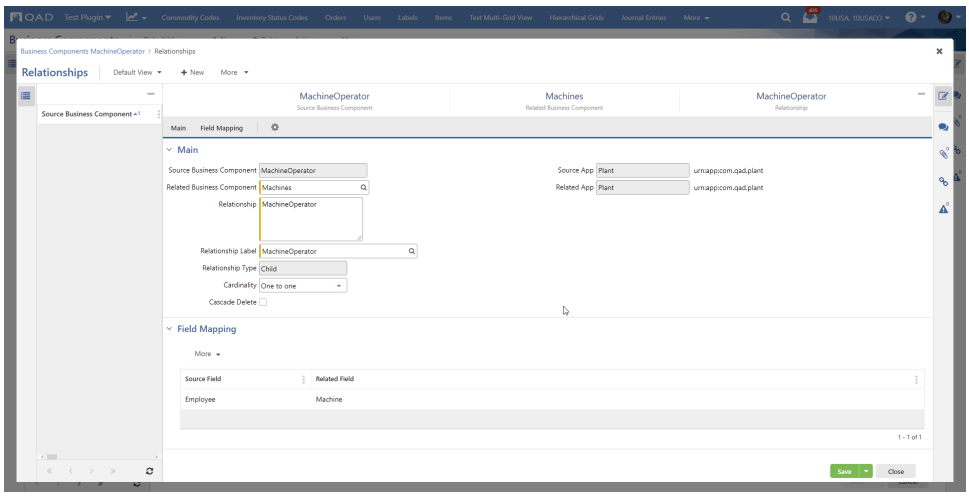
Step 2 - Create a relationship between the embedded BC and the Main Platform BC

Note: This relationship can be created or edited only on the embedded Business Component side.

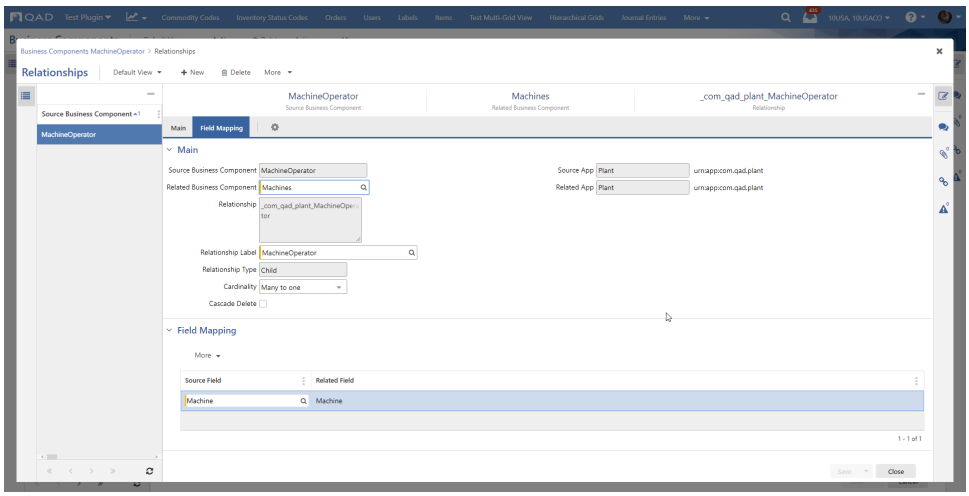
1. Open the **Relationships** panel in the **Business Components** screen, and then click the **New** button.
2. Click the lookup in the "Related Business Component" field and, in the browse, set the criteria for the platform Main BC that is aimed to be extended. Then, click the **Search** button.
   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 relationship.

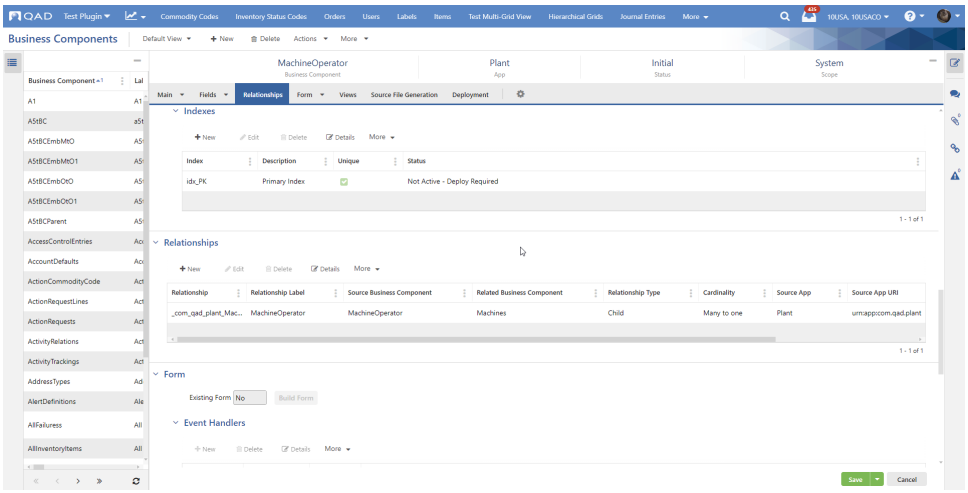


6. If you want to create a relationship with cardinality Many-to-one, change it by selecting it in the "Cardinality" drop-down list.



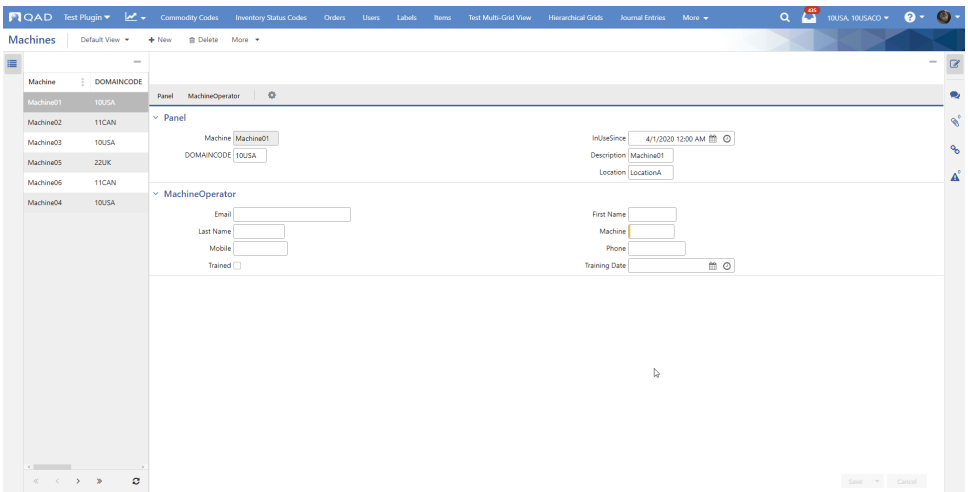
7. Click the lookup in the Source Field and select the field as Foreign Key in this Relationship. Remember that you can't use the "Primary Key" field as Foreign Key for this Relationship.
8. After that save the relationship.

9. Confirm the newly created relationship by clicking the **Save** button.

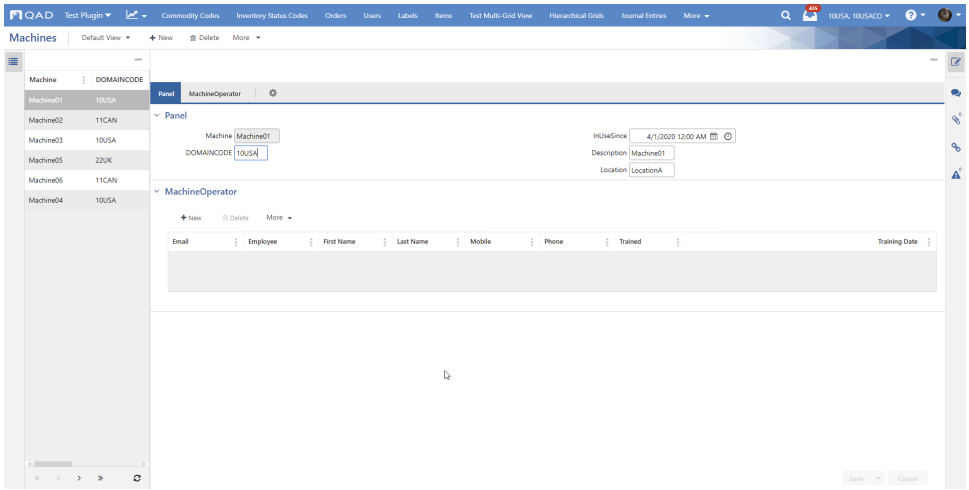


Step 3 - Deploy the embedded BC which is aimed to extend the Main Platform BC

1. Deploy the embedded Business Component. You can't use data import for the deployment of the embedded Business 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 autogenerated 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.

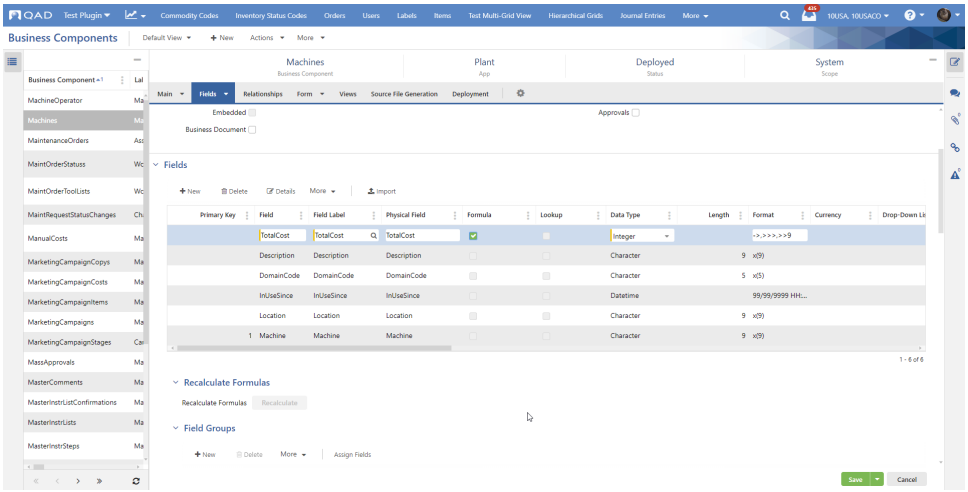


5. The embedded BC is ready for filling with data.

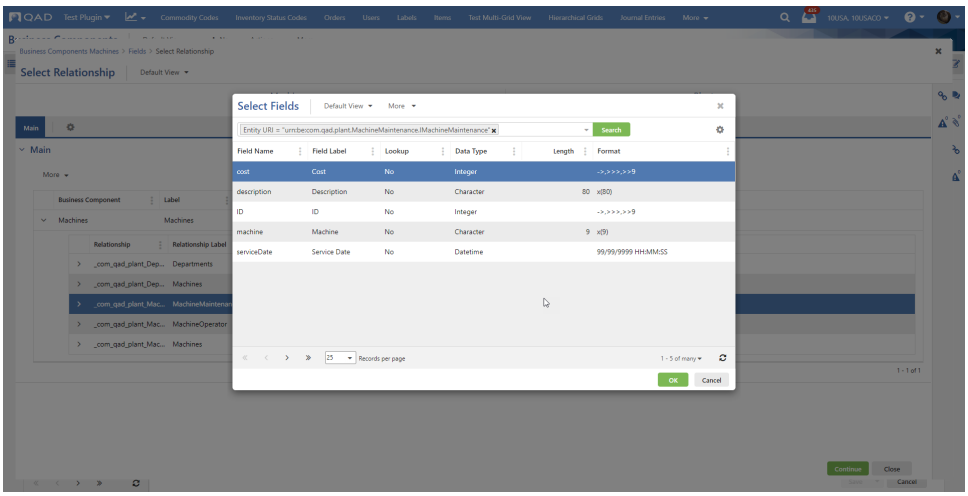
8. Extend a business component with formulas

In order to extend a business component with formulas, you need to do the following:

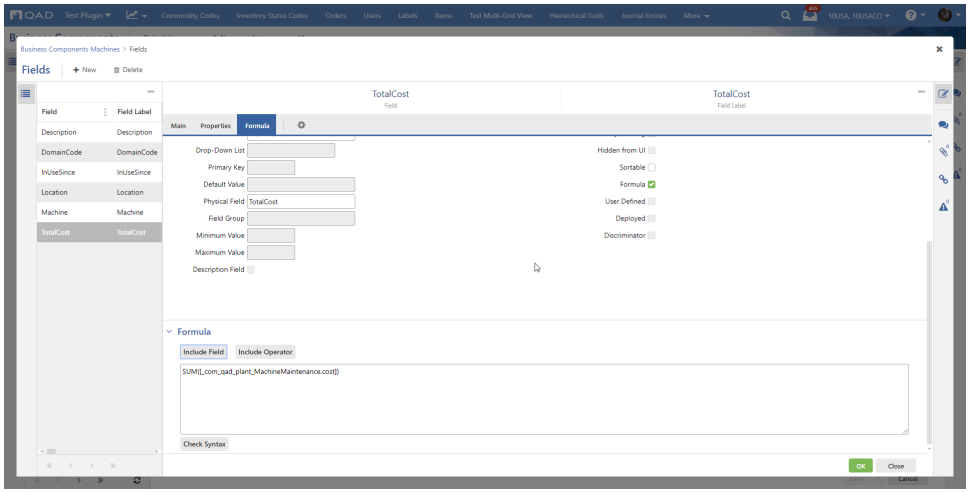
1. Navigate to **Business Components** from the menu search and find the Business Component you want to extend with formula fields.
2. Click the **Edit** toolbar button.
3. On the **Fields** panel, click the **New** button, and then fill all necessary characteristic of the field.
4. Select the checkbox for the **Formula** column.



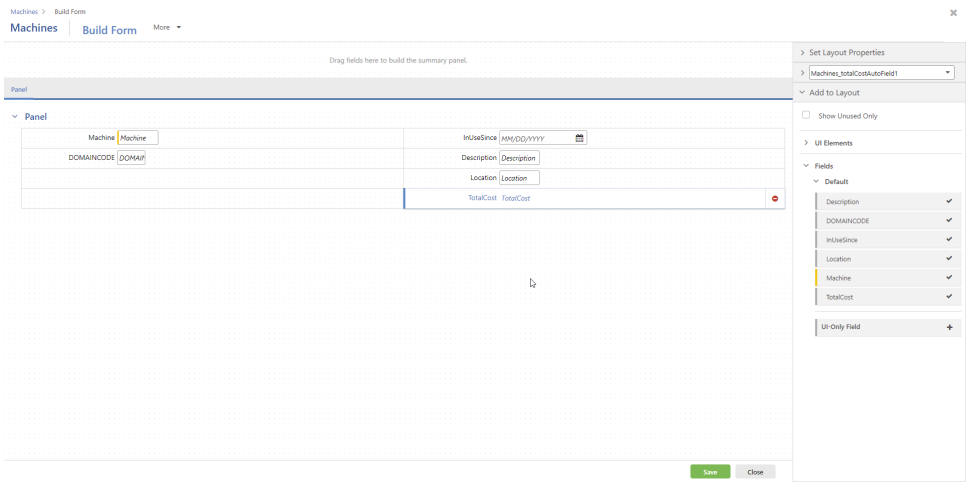
5. On the **Fields** panel, click the **Details** button, and then scroll down to the **Formula** panel.
6. Change the formula definition. If you need some other field to be involved in the calculation, use the **Include Field** button. Use the **Include Operator** button to choose operators needed for the calculation.
   In this example, the additionally created embedded BC MachineMaintenance, which is extending the Main Platform BC Machines (Many-to-one), is used.



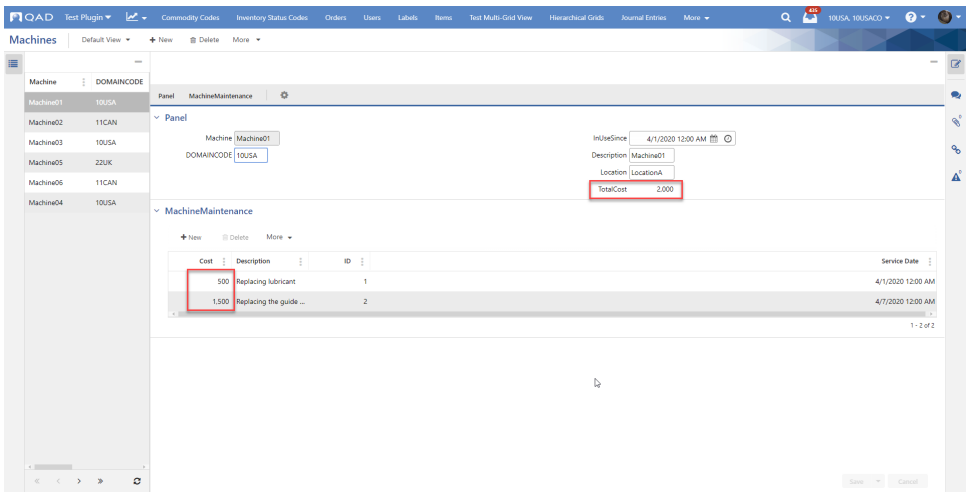
7. Click the **Check Syntax** button to verify if the formula definition is correct. You can save the formula definition only after checking the formula syntax.



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.



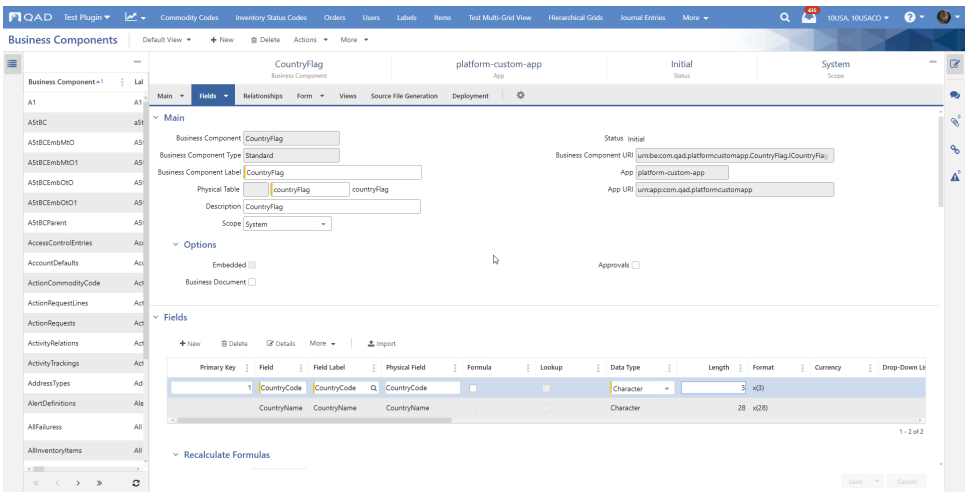
11. Open a Hybrid Browse of the Business Component using the menu search to see the result.



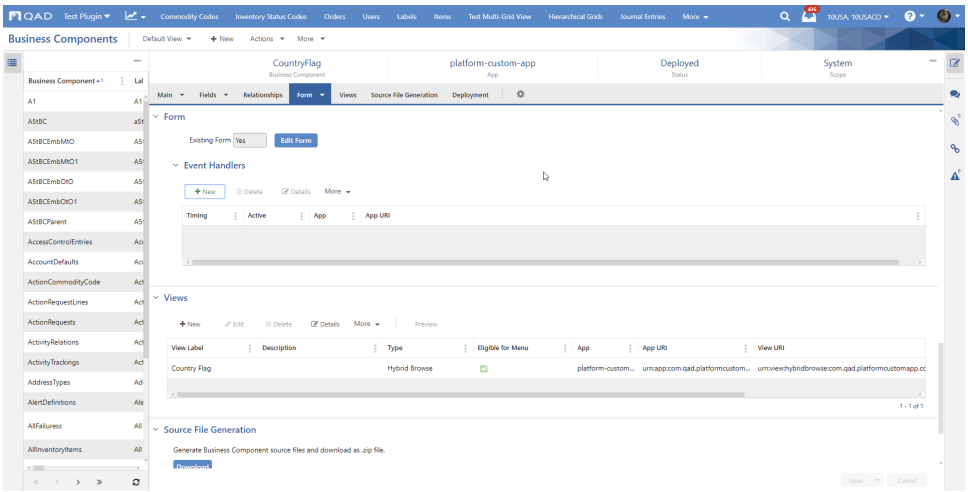
9. Extend a business component UI with event handlers

In order to extend a business component UI with event handlers, you need to do the following:

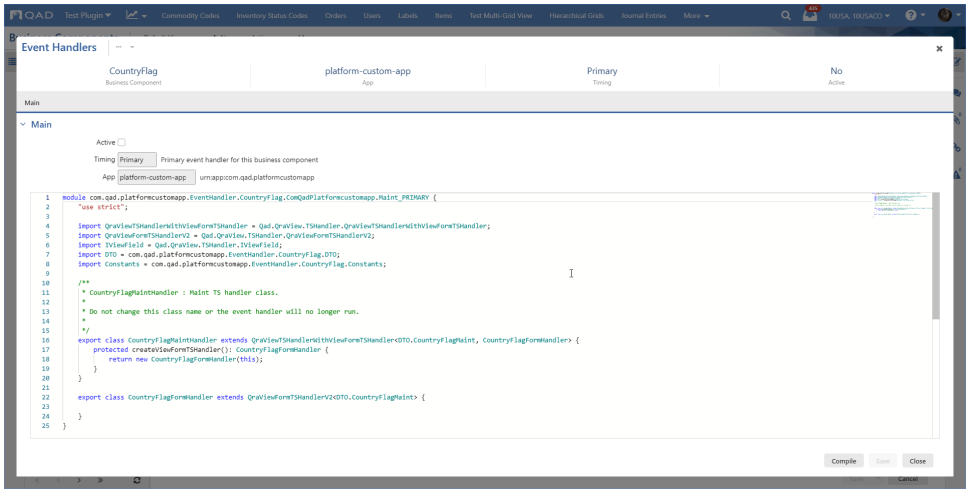
1. Create a Business Component and create a Form and a View/Hybrid Browse for it, and then deploy the Business Component.



2. Go to the **Form** panel of the Business Component created in the 1st step.

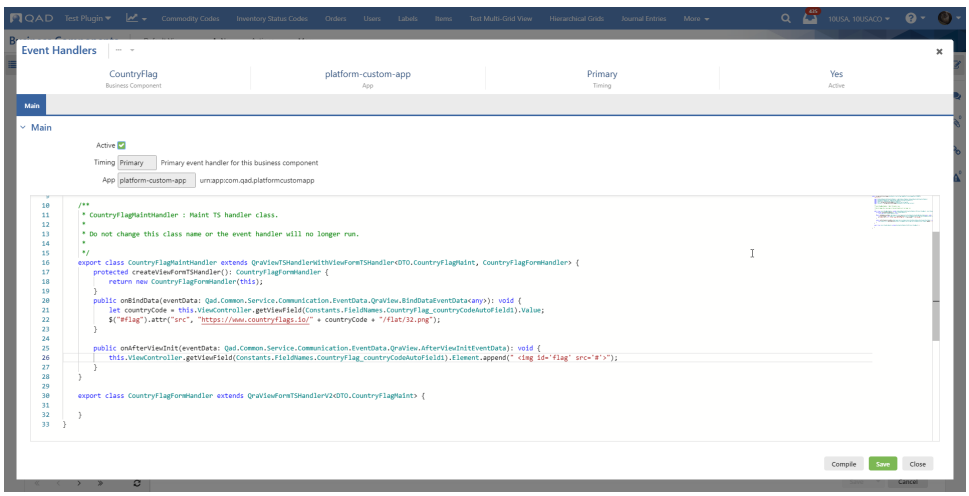


3. Click the **New** button on the **Event Handlers** panel. You will see the following pop-up window with prepared TypeScript template.

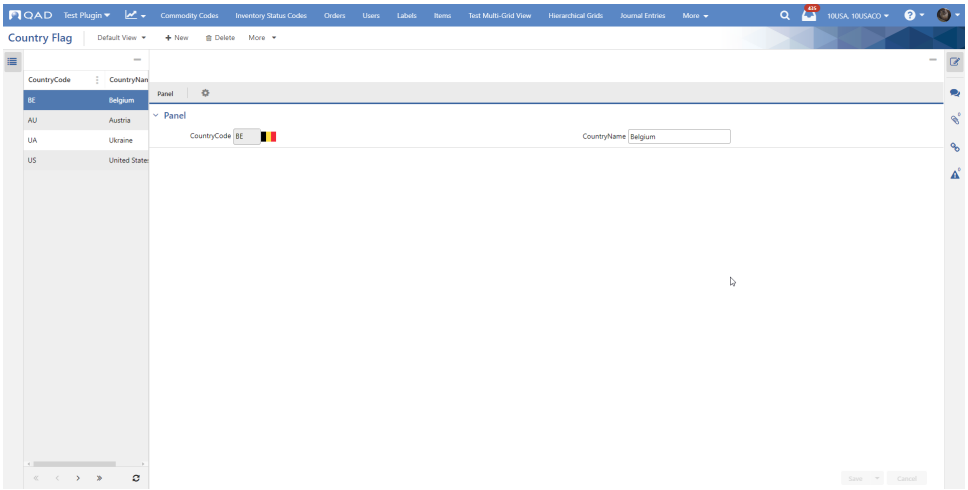


4. Select the **Active** checkbox.
5. Change TS Code according to the expected behavior. For example, paste the following code on line 21:

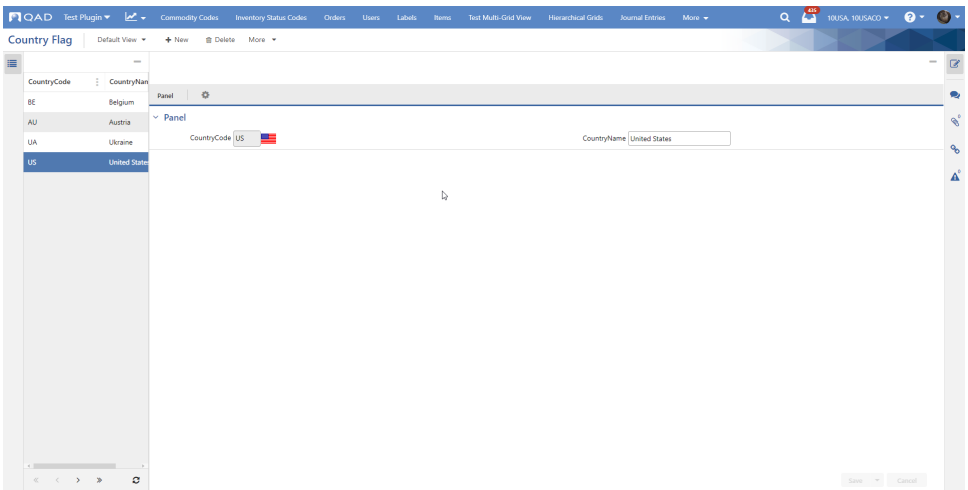
```
public onBindData(eventData: Qad.Common.Service.Communication.EventData.QraView.
BindDataEventData<any>): void {
        let countryCode = this.ViewController.getViewField(Constants.FieldNames.
CountryFlag_countryCodeAutoField1).Value;
    $("#flag").attr("src", "https://www.countryflags.io/" + countryCode + "/flat/32.png");
}

public onAfterViewInit(eventData: Qad.Common.Service.Communication.EventData.QraView.
AfterViewInitEventData): void {
        this.ViewController.getViewField(Constants.FieldNames.
CountryFlag_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 and add records.



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 platform BCs.

The following page explains the mechanism that support all capabilities: Extending the OOABL Business Logic

Steps to take for the OOABL coding

Platform Development in YAB

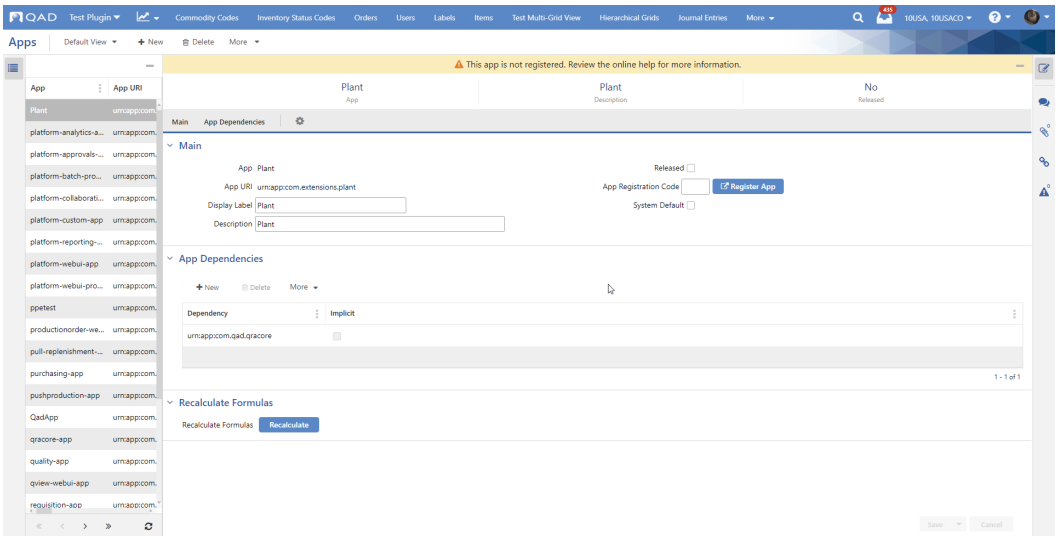
Use cases for extending OOABL code

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 Action Centers

In item 12, you will create KPIs and an Action Center. When Action Centers and KPIs are created by users, they are automatically associated with the default app for the environment. Those Action Centers and KPIs are automatically exported with that app using the YAB *app-export* command.

Note: When you migrate a KPI from a source environment to a target environment, the migration does not include the browse that is used as the data source. You must ensure the browse used as the data source already exists in the target environment and that it has the same fields in its definitions. You can migrate browse definitions with the import/export tool in Browse Maintenance in the QAD .NET UI.

Export App

To export all resources, do the following:

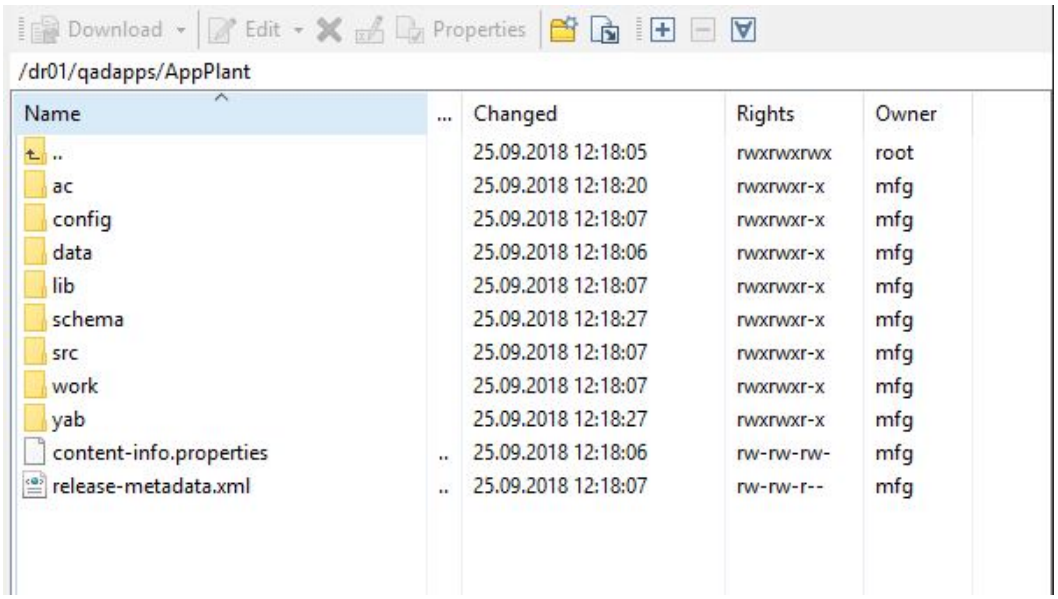
1. Connect by SSH to the current environment.
2. Navigate to the working folder in the current environment.
3. Run the YAB command:
   yab app-export -dir:*<directory to export to>* **-appuri:***<app uri>*

   In our example, that is:
   yab app-export -dir:/dr01/qadapps/AppPlant -appuri:urn:app:com.extensions.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 -dir:/dr01/qadapps/AppPlant

                      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)
```

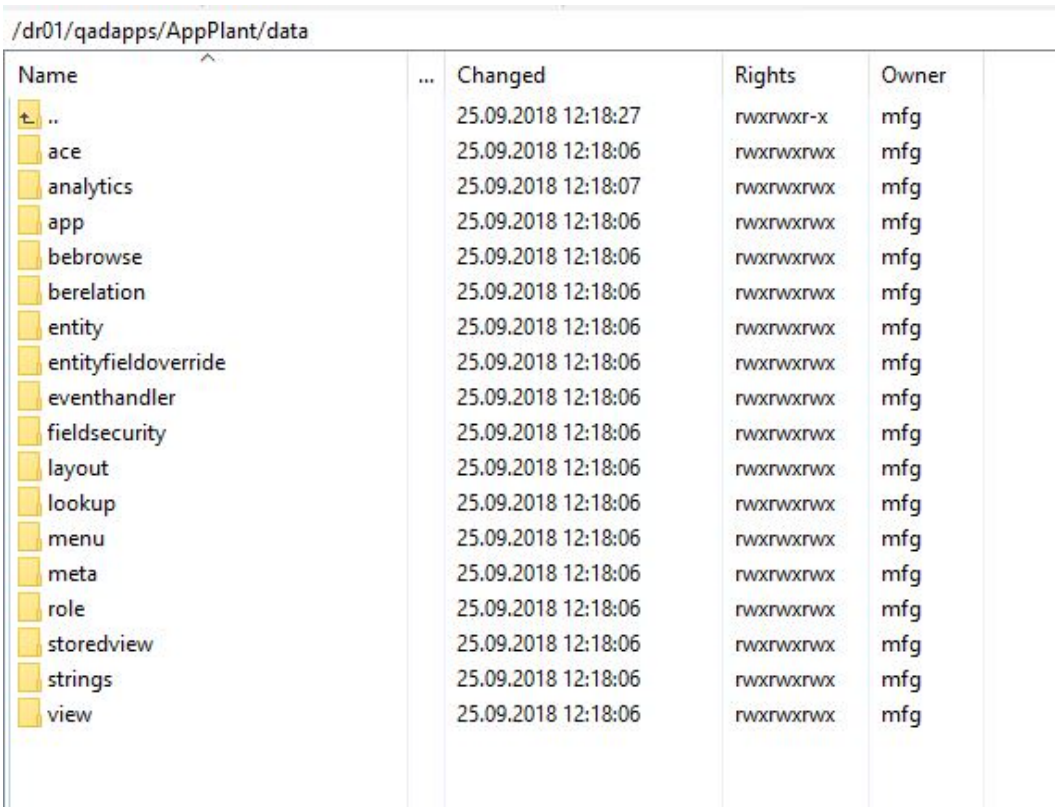
Note: For detailed information, see Platform Development in YAB.

The target folder contains all related information (described in xml files) about the exported App, including earlier created secured resources, schema dump, and other.



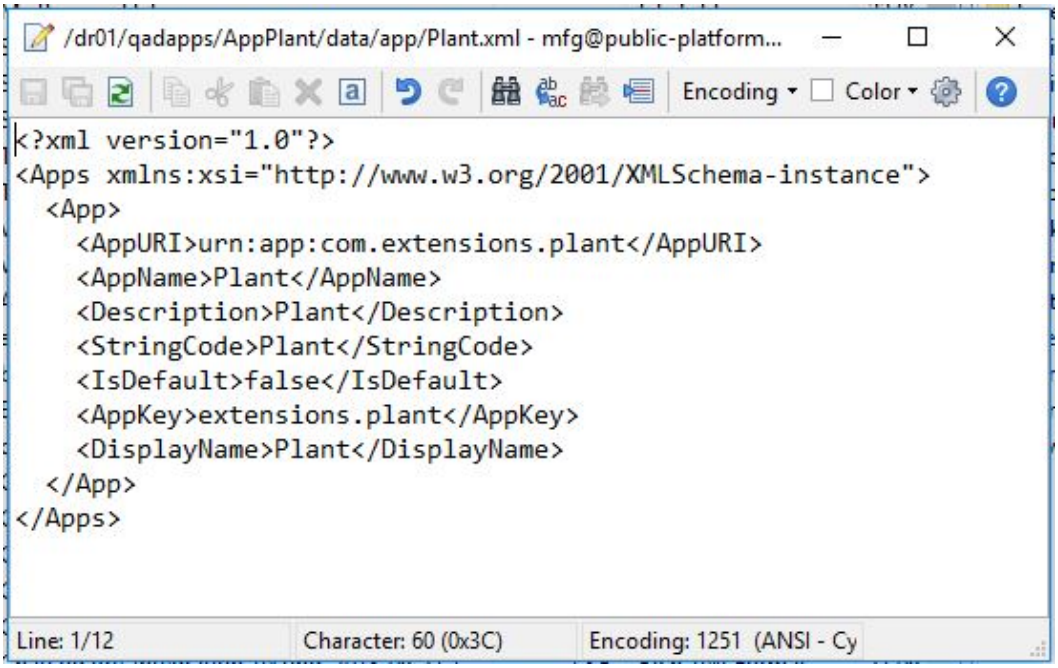
Name	...	Changed	Rights	Owner
..		25.09.2018 12:18:05	rwxrwxrwx	root
ac		25.09.2018 12:18:20	rwxrwxr-x	mfg
config		25.09.2018 12:18:07	rwxrwxr-x	mfg
data		25.09.2018 12:18:06	rwxrwxr-x	mfg
lib		25.09.2018 12:18:07	rwxrwxr-x	mfg
schema		25.09.2018 12:18:27	rwxrwxr-x	mfg
src		25.09.2018 12:18:07	rwxrwxr-x	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:

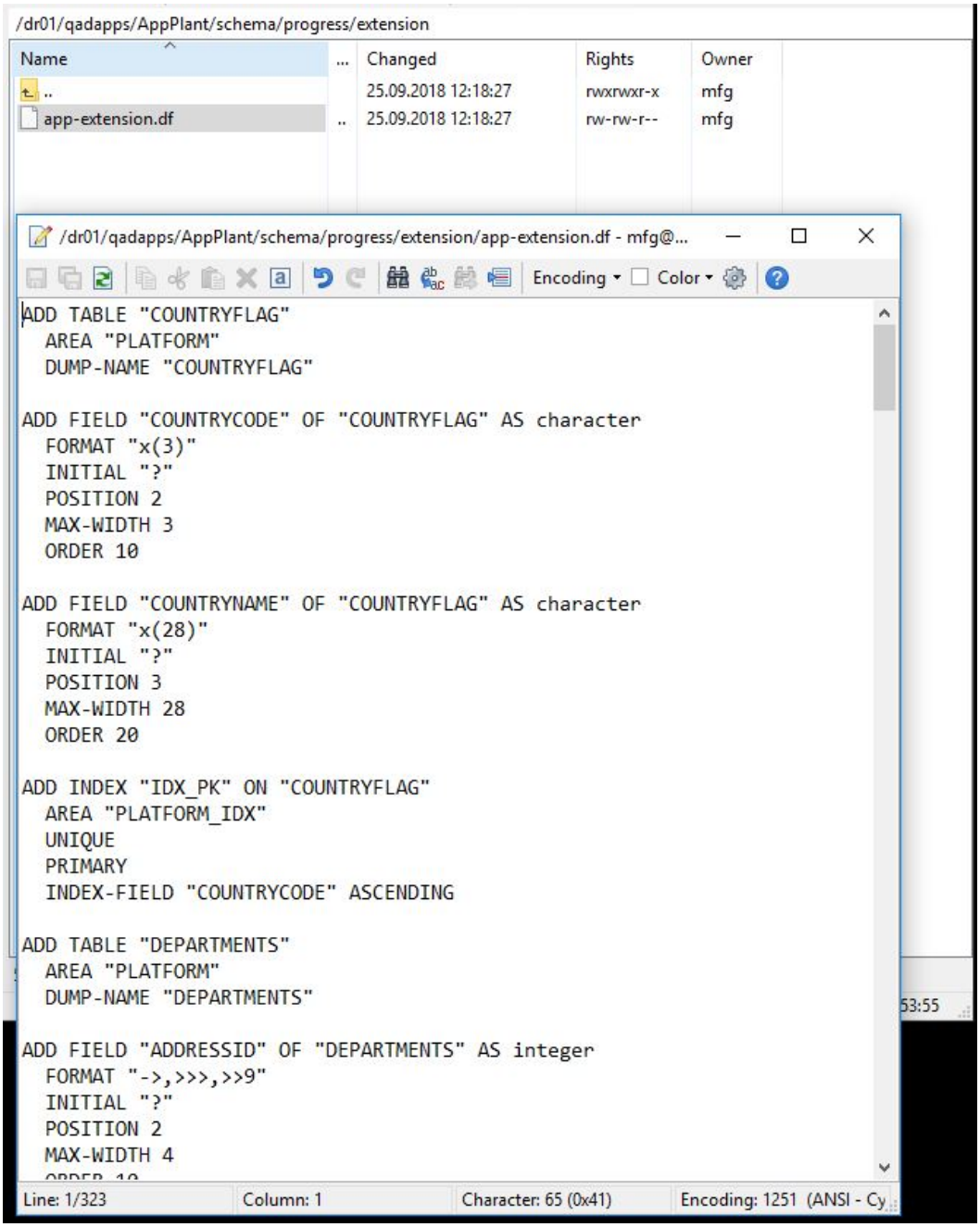


Name	...	Changed	Rights	Owner
..		25.09.2018 12:18:27	rwxrwxr-x	mfg
ace		25.09.2018 12:18:06	rwxrwxrwx	mfg
analytics		25.09.2018 12:18:07	rwxrwxrwx	mfg
app		25.09.2018 12:18:06	rwxrwxrwx	mfg
bebrowse		25.09.2018 12:18:06	rwxrwxrwx	mfg
berelation		25.09.2018 12:18:06	rwxrwxrwx	mfg
entity		25.09.2018 12:18:06	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 a package

1. Connect by SSH to the current environment.
2. Navigate to the working folder of current environment.
3. Run the 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 -version:1.0.0.0

                  system-package-create (1 task)                        [APPLY]
------------------------------------------------------------------------------
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

A 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

Install the package 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 the YAB command: **yab install** *<name of package with version and extension>*

```
BUILD SUCCESSFUL (1.319 s)
[mfg@patformbranch final]$ yab install appplant-1.0.0.0.zip
```

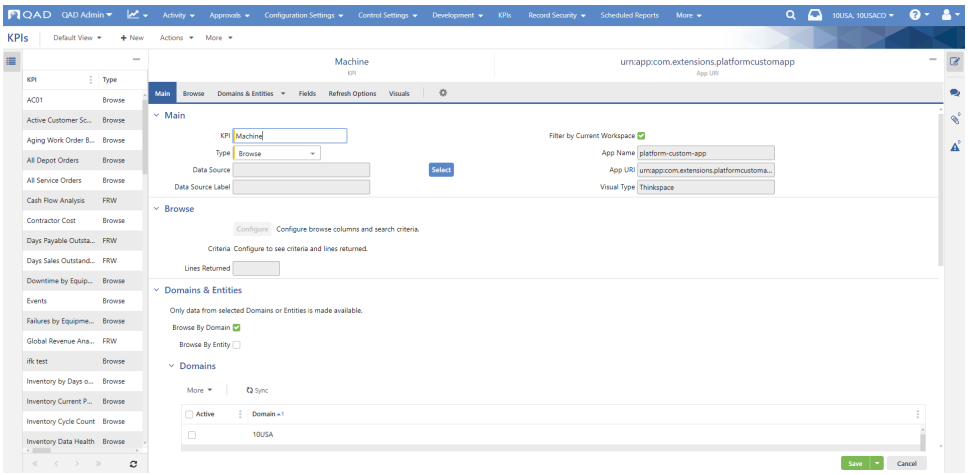
Installing the package can take the same time as updating the environment.

After successful installation of the package, the target environment will contain all the resources and information.

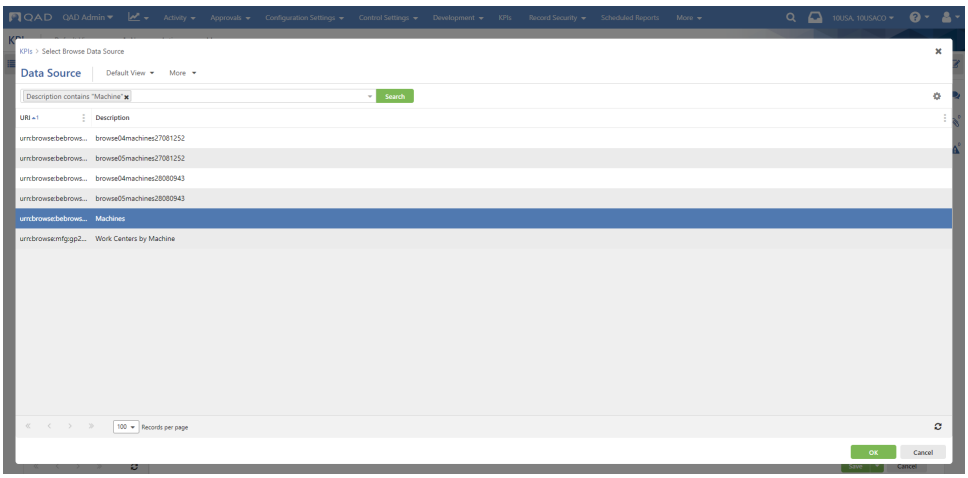
Note: For detailed information, see Platform Development in YAB.

12. Create KPIs

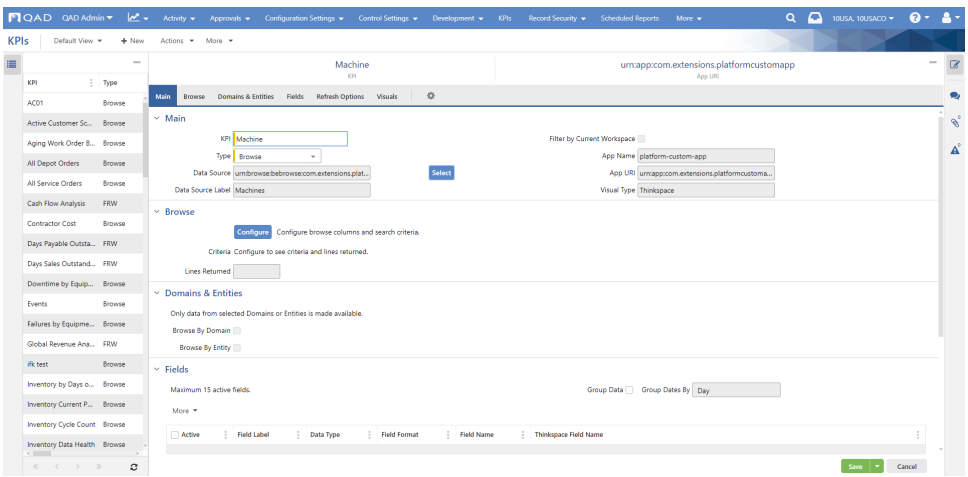
1. Navigate to **KPIs** from the menu search.
2. Click the **New** toolbar button.
3. Complete the **KPI** field.



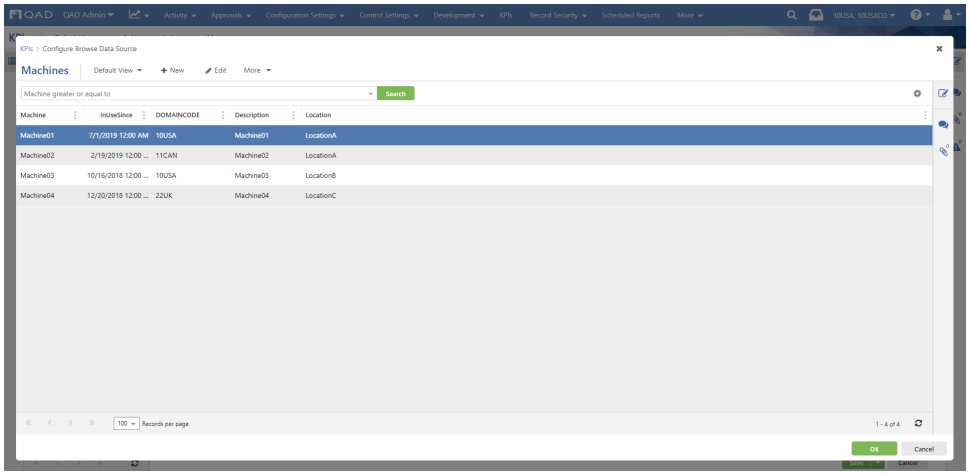
4. Click the **Select** button and choose a business component's browse.



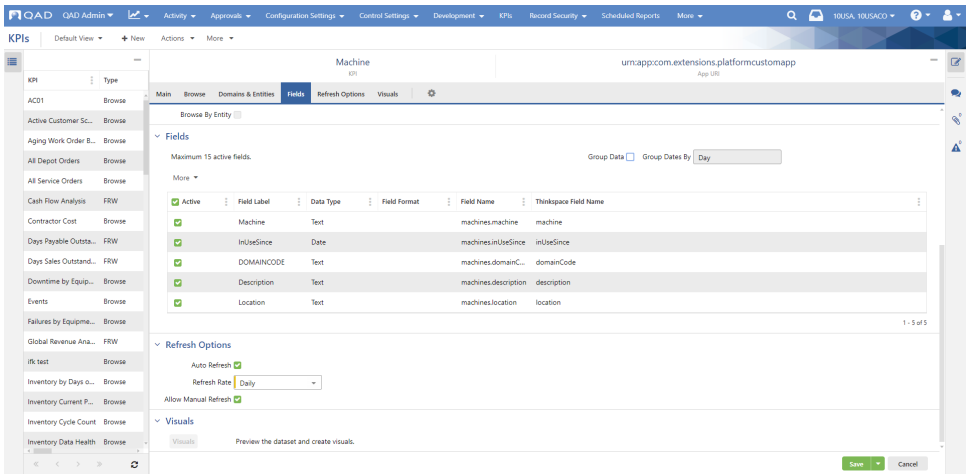
5. Click **OK**.



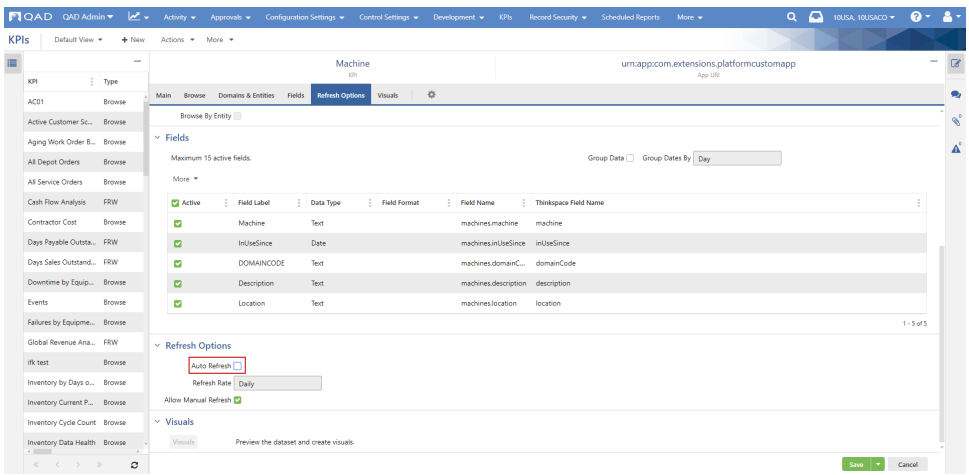
6. Click the **Configure** button on the **Browse** panel to configure browse columns and search criteria.



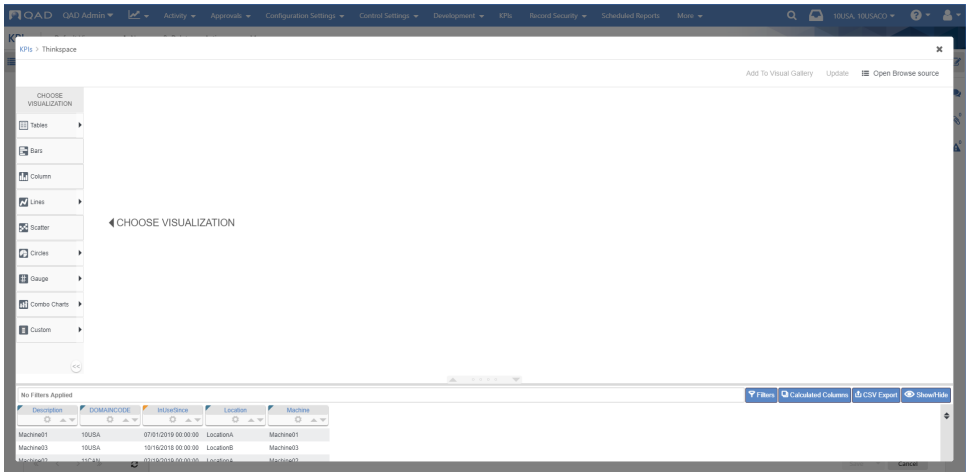
7. Click **OK** and go to the **Fields** panel. The fields from Browse are included.



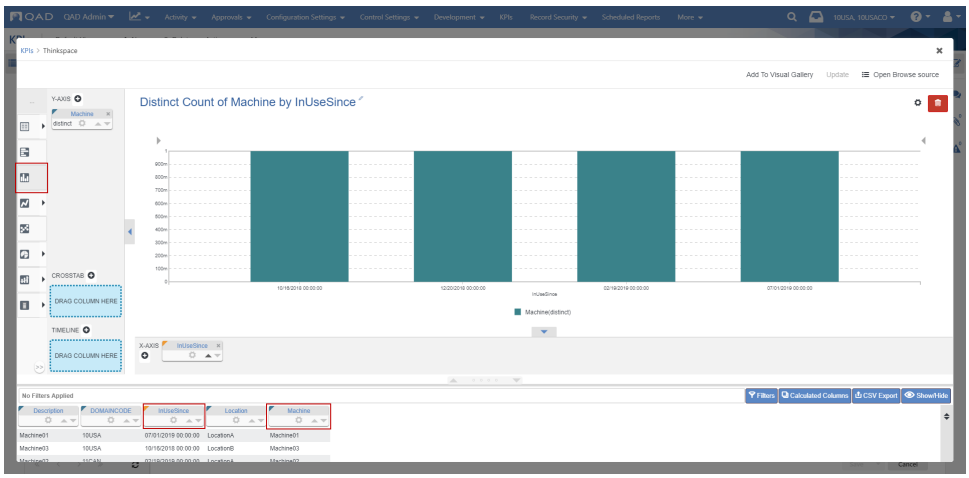
8. On the **Refresh Options** panel, clear the "Auto Refresh" option to prevent exceeding the limit of the auto-refreshed KPIs.



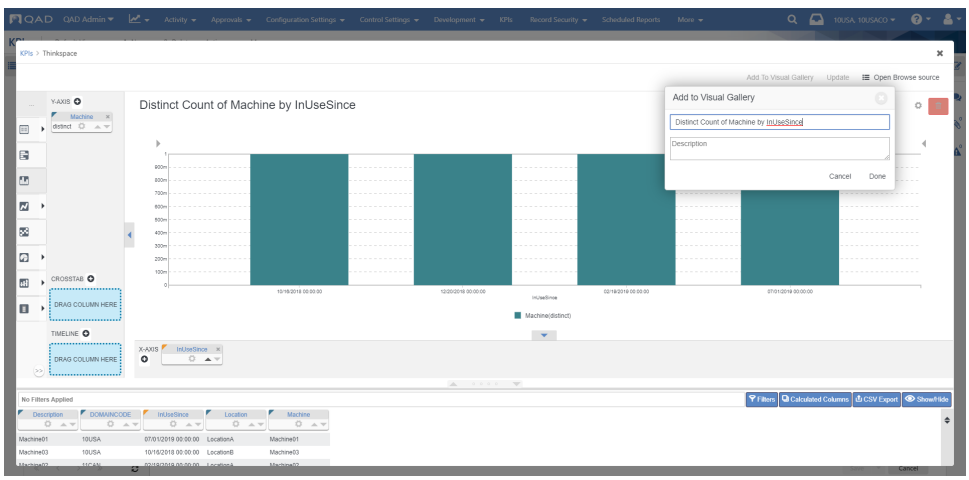
9. Save a new KPI.
10. On the **Visuals** panel, click the **Visuals** button to create visuals.



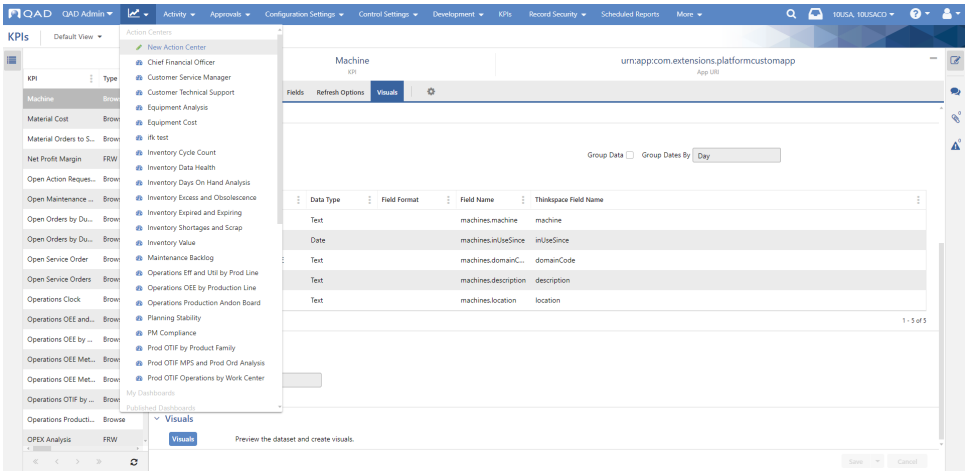
11. Choose the **Columns** visualization on the left. Drag Machine to the Y-AXIS and InUseSince to the X-AXIS.



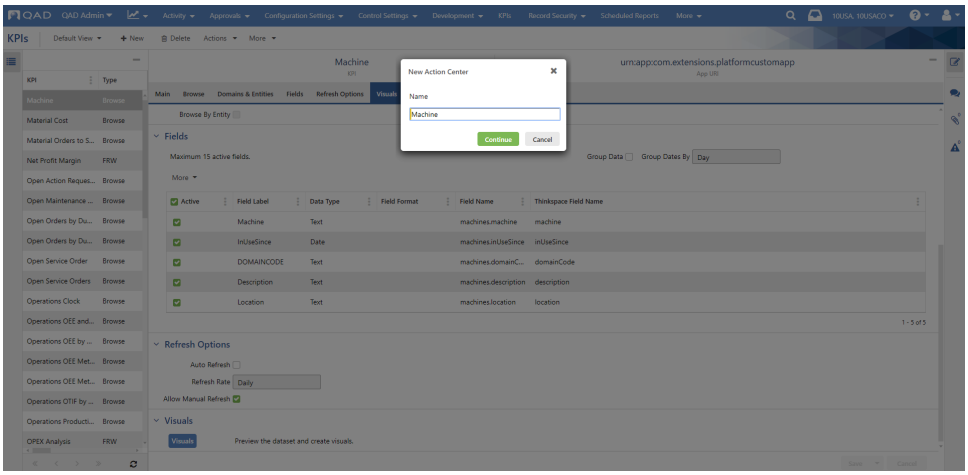
12. Click **Add to Visual Gallery** and enter the name of your visual and description (if needed), and then click **Done**.



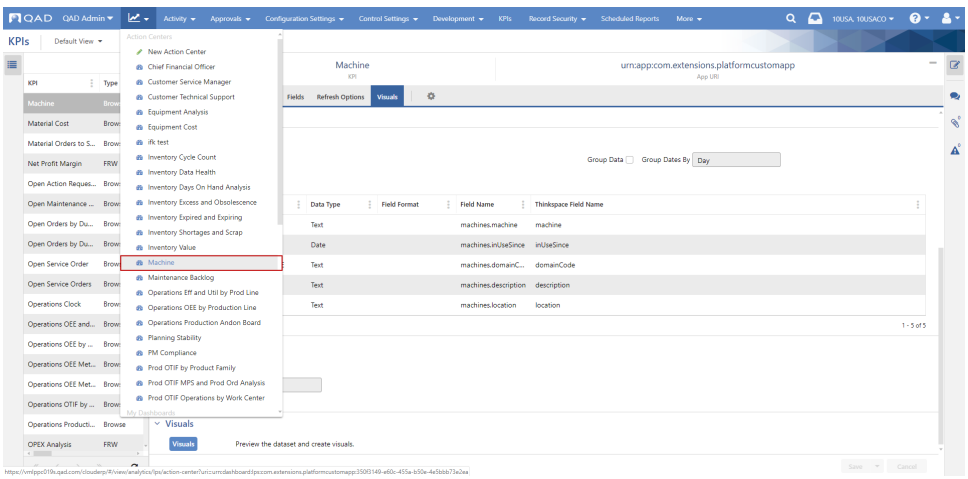
13. Click **Menu Dashboard** and choose **New Action Center**.



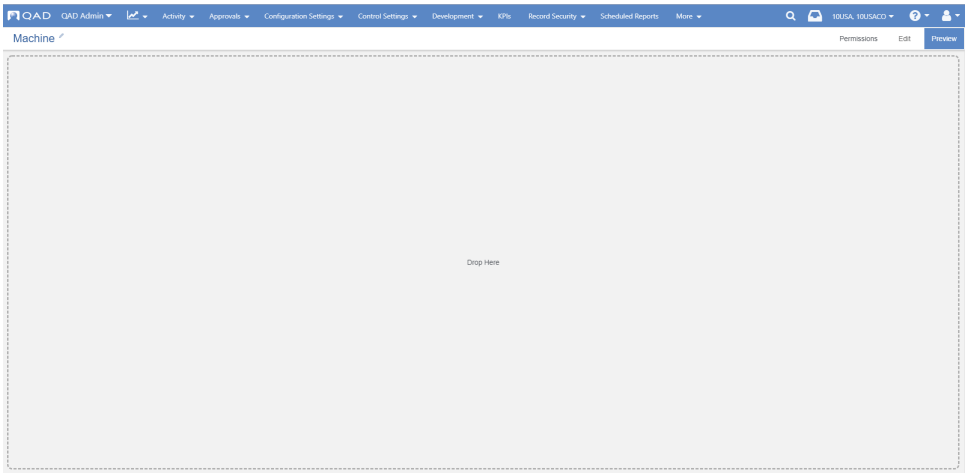
14. Define the name for a new Action Center and click **Continue**.



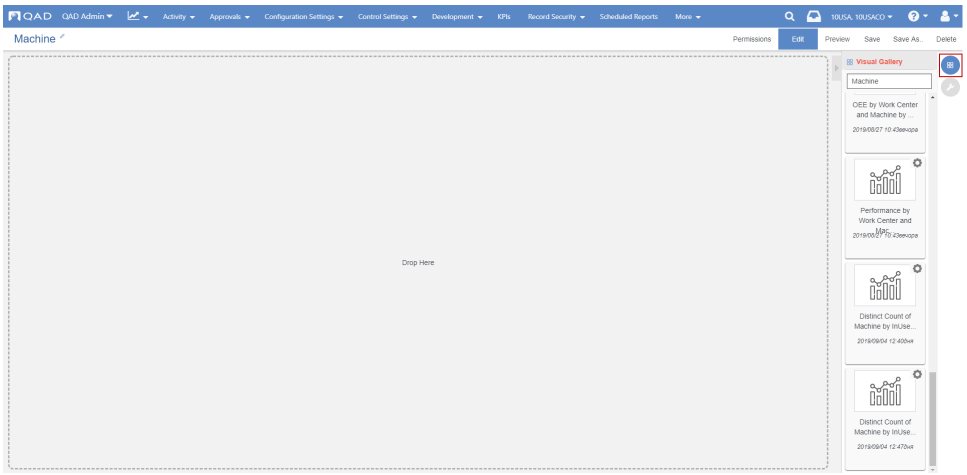
15. Click **Menu Dashboard** and choose the created Action Center.



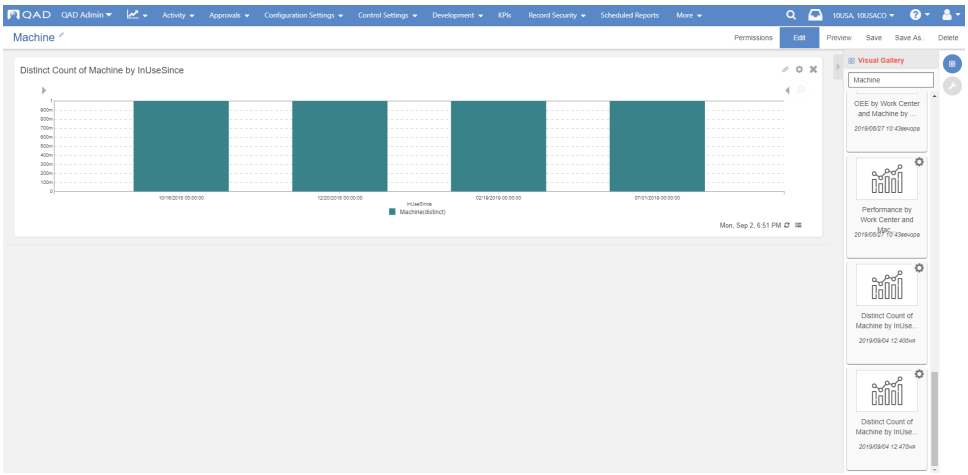
16. A new Action Center opens.



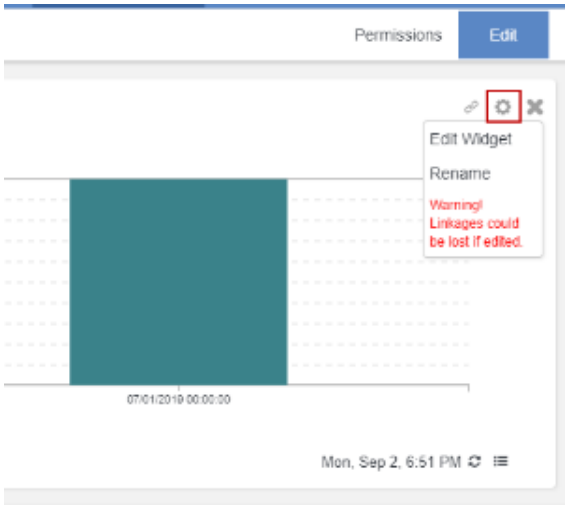
17. Click **Edit** at the top-right corner of the screen. Then click the **Visual Gallery** icon and search for the last created visual. Note that the search is case-sensitive.



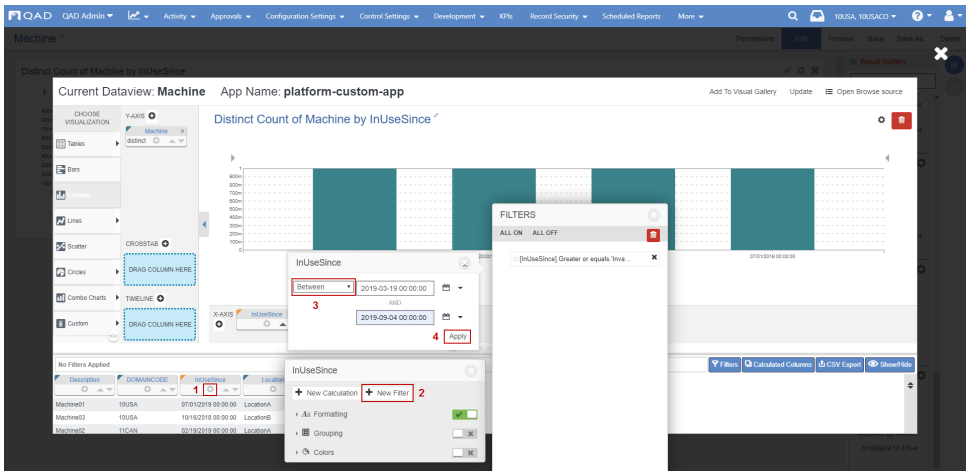
18. Drag-and-drop the visual on the empty screen.



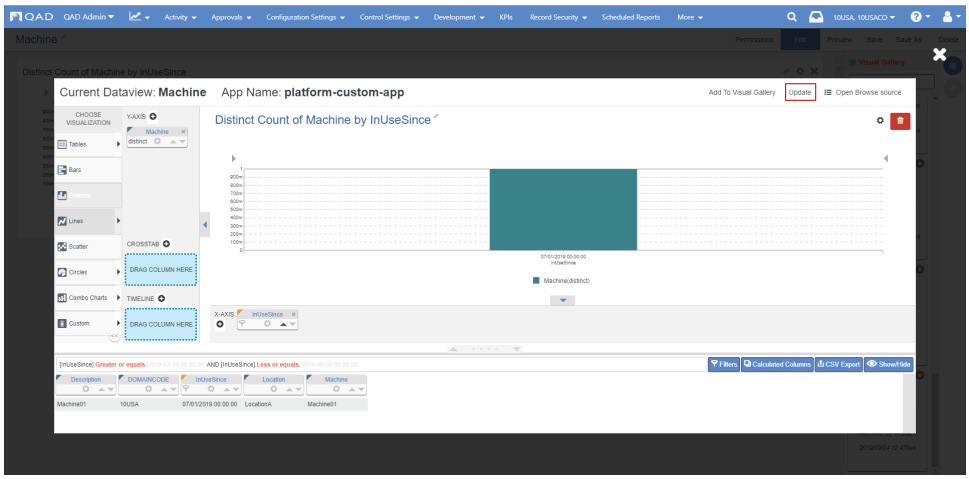
19. Click the gear icon in the top-right corner and select **Edit Widget** to edit the existing chart.



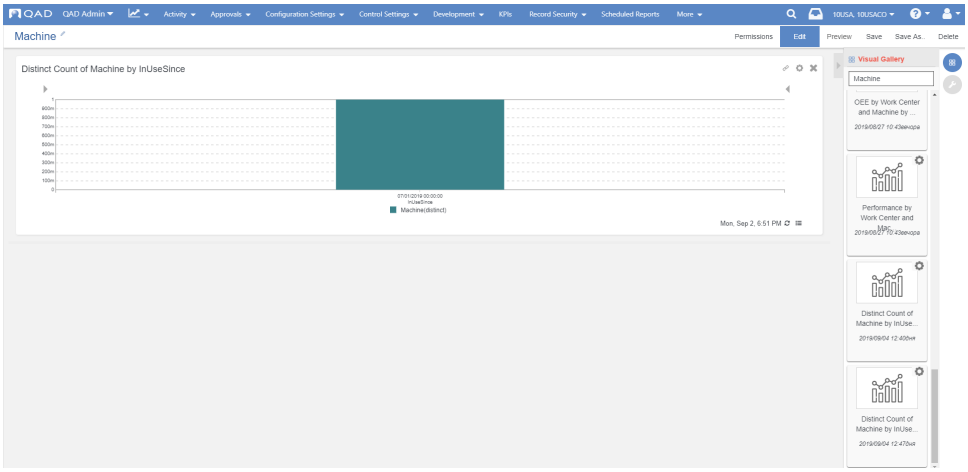
20. Click the gear icon in the InUseSince column, and then click **+ New Filter**. Set condition "Between" and select the dates from the calendar. Click **Apply**.



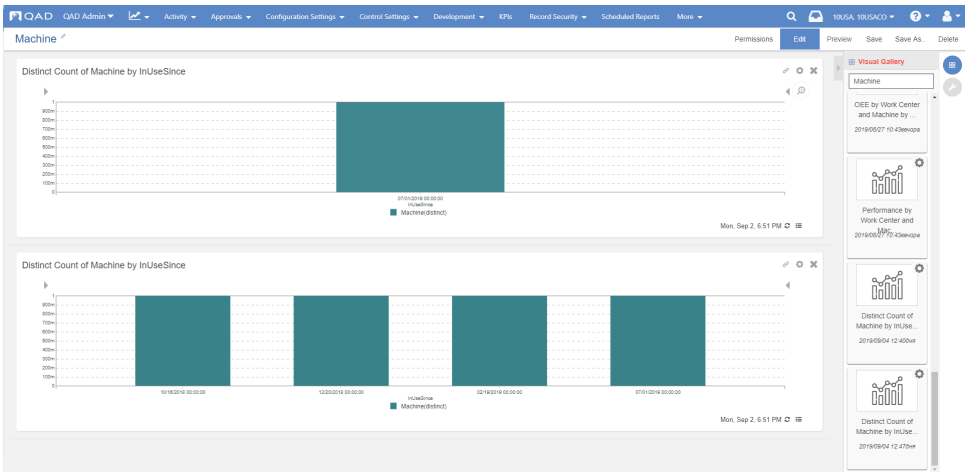
21. Click the **Update** button and close the window.



22. Now the edited visual displays in the Action Center.



23. Search for the first created visual and drop it on the screen.
24. 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:

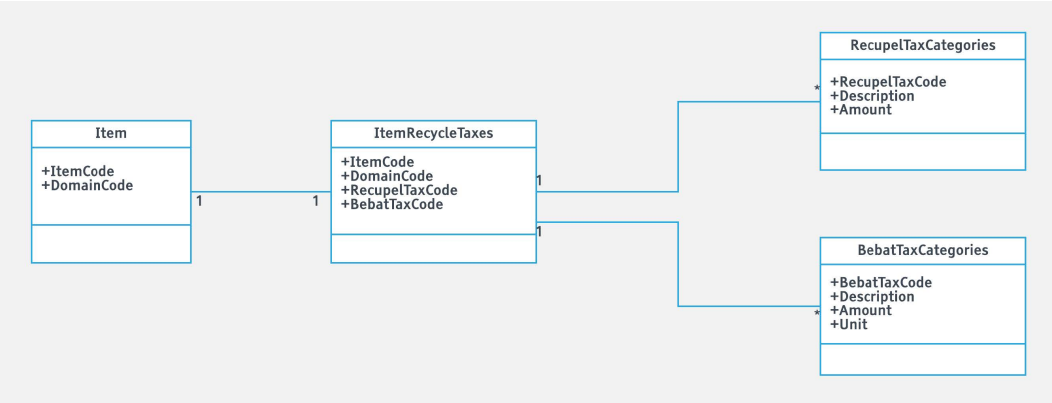
- Extend the UI with extra fields about recycle tax information.
- Extend the UI with extra fields and a grid about electrical compatibility information.

Steps to follow:

Example 1.1: Adding new functionality to item maintenance with a One-to-one relationship

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 One-to-one relationship between Item and ItemRecycleTaxes is a child-parent relationship (because we want to add extra fields on the items UI). The 1-N relationships to RecupelTaxCategory and BebatTaxcategory are lookup relationships. They make sure that we will see lookups on the fields that we are adding to items. See Business Component Relationship for more info on the different types of relationships.

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 Components
- 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 OOABL validation to check if a tax category is still referenced when deleting (using IVirtualBusinessEntity)
- Ref: complete event handler code for the Items BC

Create RecupelTaxCategory Components

This section explains all the steps to create the Recupel Tax Category business component:

- Create RecupelTaxCategories Business Component
- Create RecupelTaxCategories View
- Create RecupelTaxCategories Browse
- Deploy RecupelTaxCategories Business Component

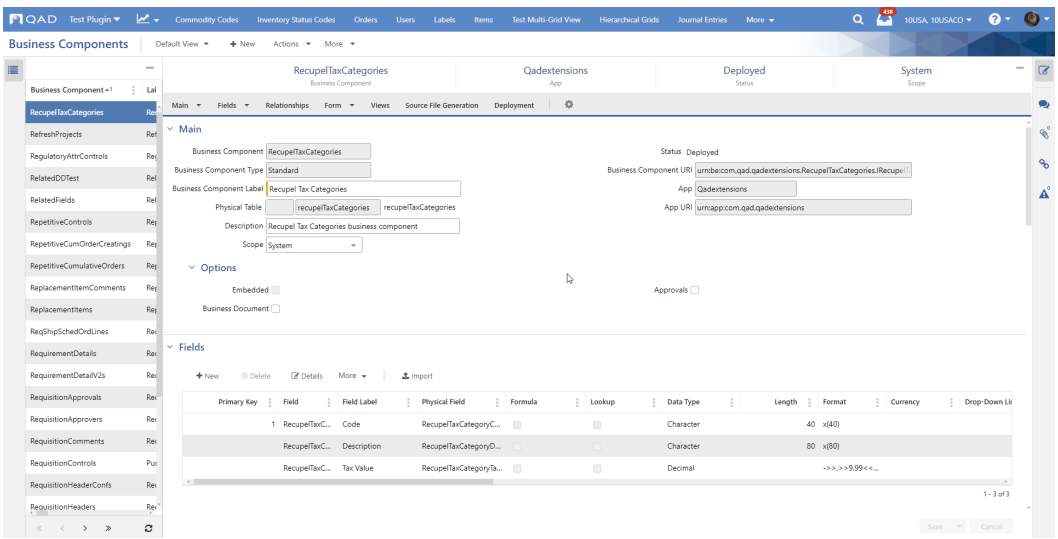
Create RecupelTaxCategories Business Component

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 database: 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**.

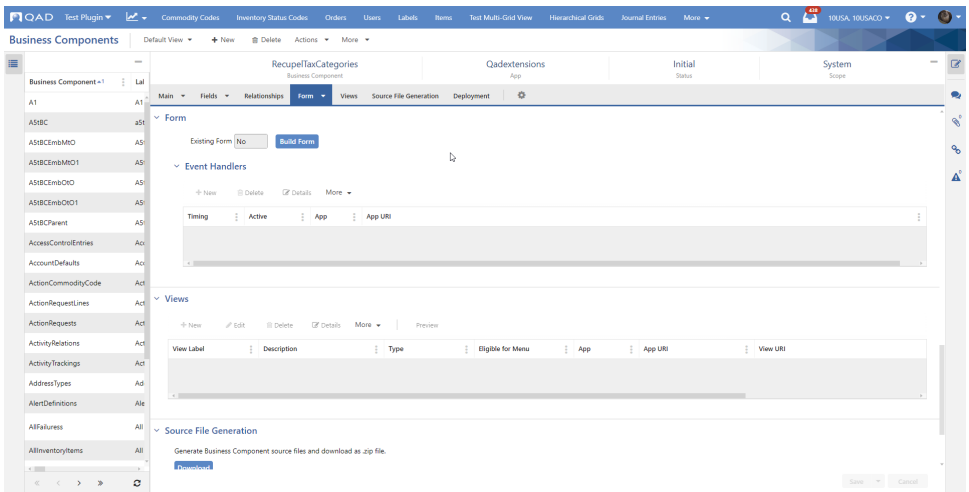
This is what the BC should look like:



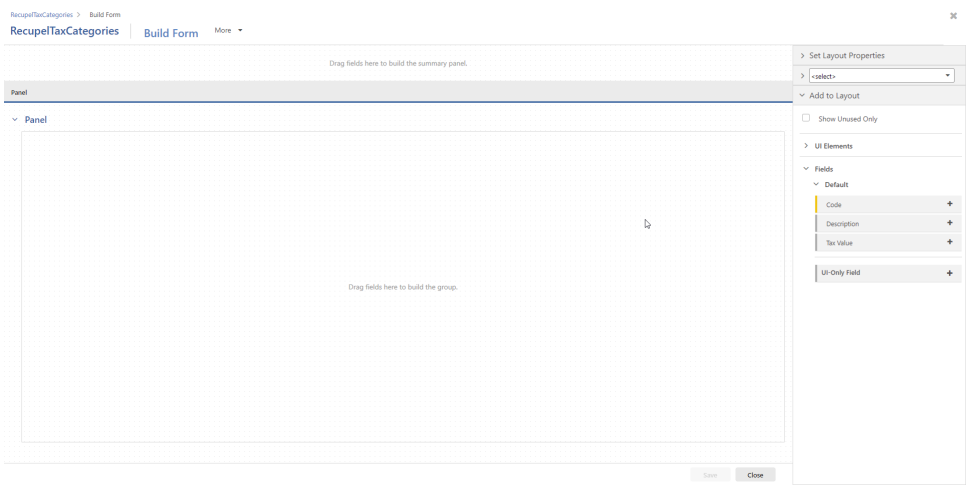
Create RecupelTaxCategories View

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:

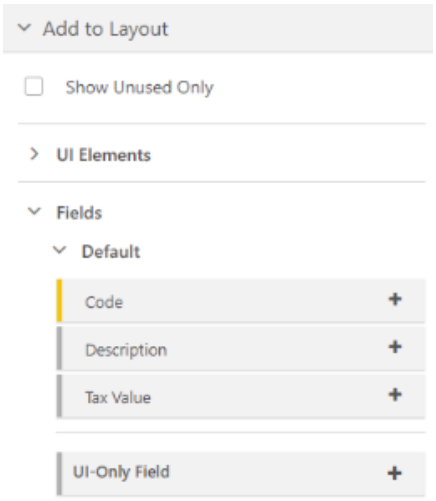
1. Open the **RecupelTaxCategories** BC.
2. Open the **Form** panel.



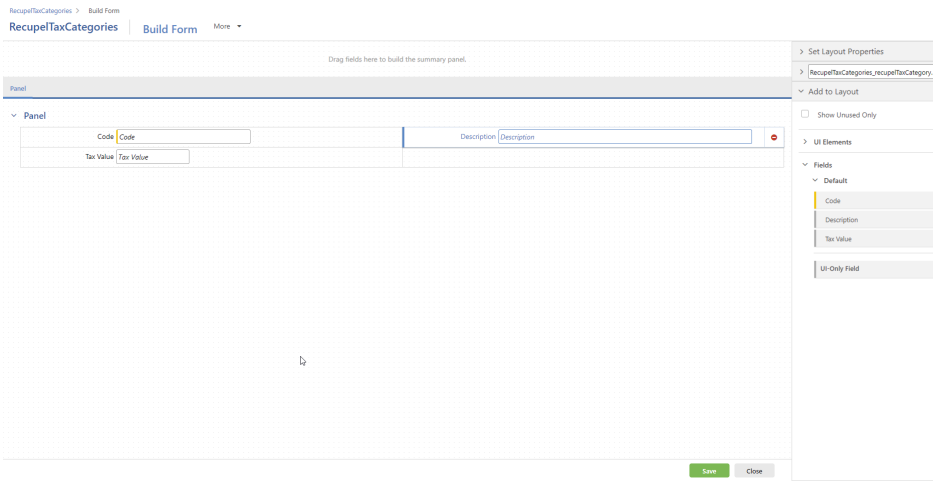
3. Click the **Build Form** button. This will open the form builder.



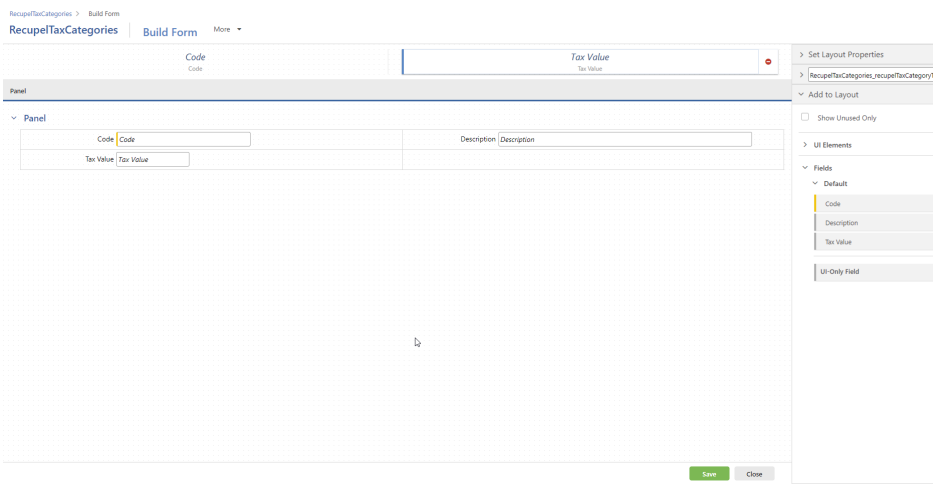
4. We now have a view with a panel that we can start dragging-and-dropping UI elements to. Follow these steps to add the elements:
    a. Click the **Fields** and **Default** menu items at the right bottom to open them.



b. Drag and drop the **Code** field on the panel.
c. Do the same for the **Description** and **Tax Value** fields so that the screen looks like this.



d. 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.

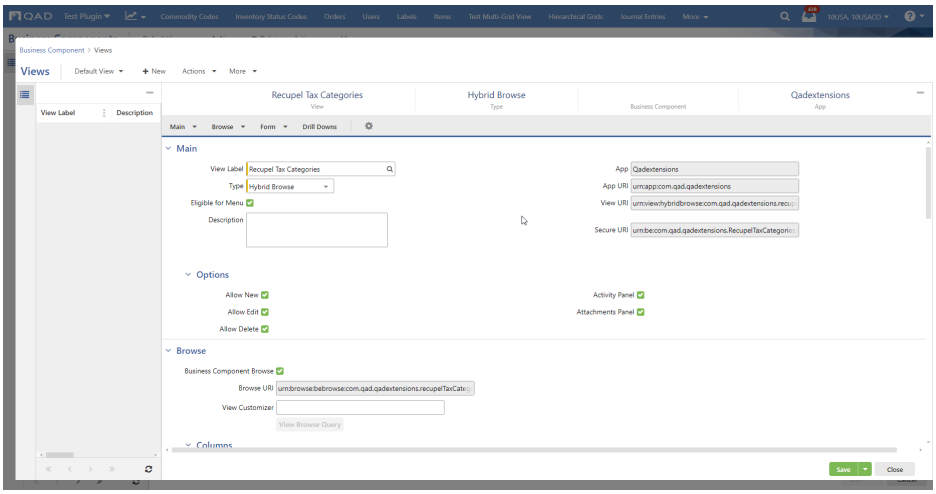


5. Now our form is ready. Click the **Save** button to save the form, and then the **Close** button.
6. After saving the form, we are back on the BC screen ready to create a Hybrid view.

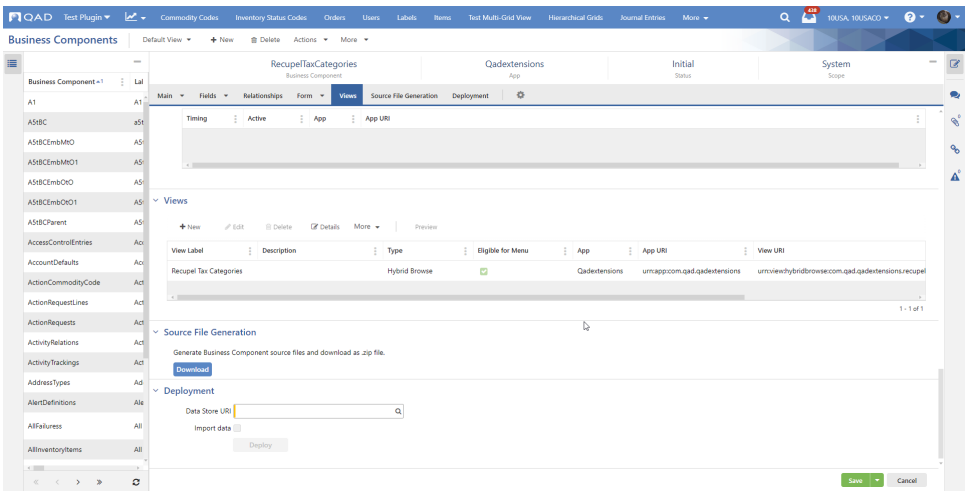
Create RecupelTaxCategories Browse

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, open 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.
    - 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.

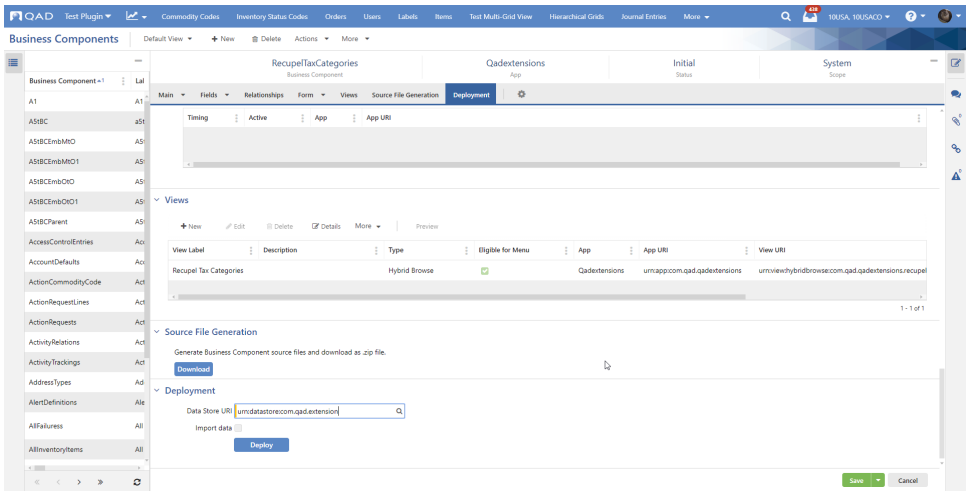


4. Now, click the **Save** button again to save changes.

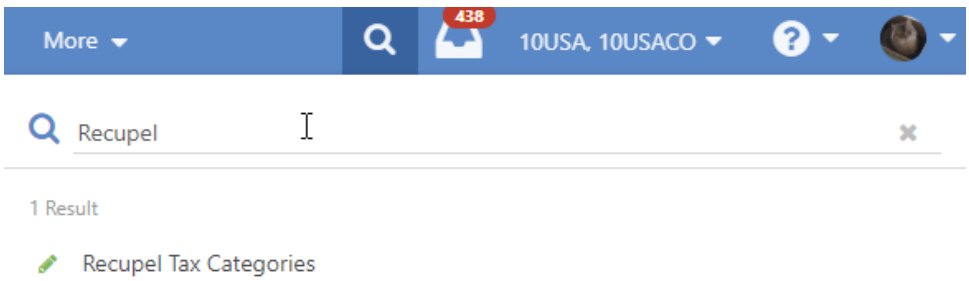
Deploy RecupelTaxCategories Business Component

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:

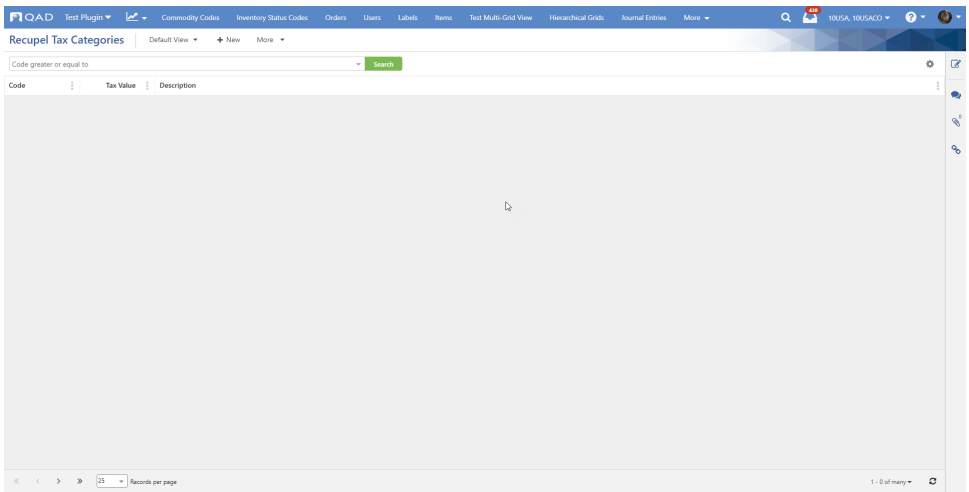
1. Open the Business Components Maintenance screen, search for the RecupelTaxCategories business component, and then click the **Edit** button.
2. Open the **Deployment** panel, and then click the lookup button on the **Data Store URI** field. This will open a lookup with the available data stores. Select 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 Components

This section explains all the steps to create the Bebat Tax Category business component:

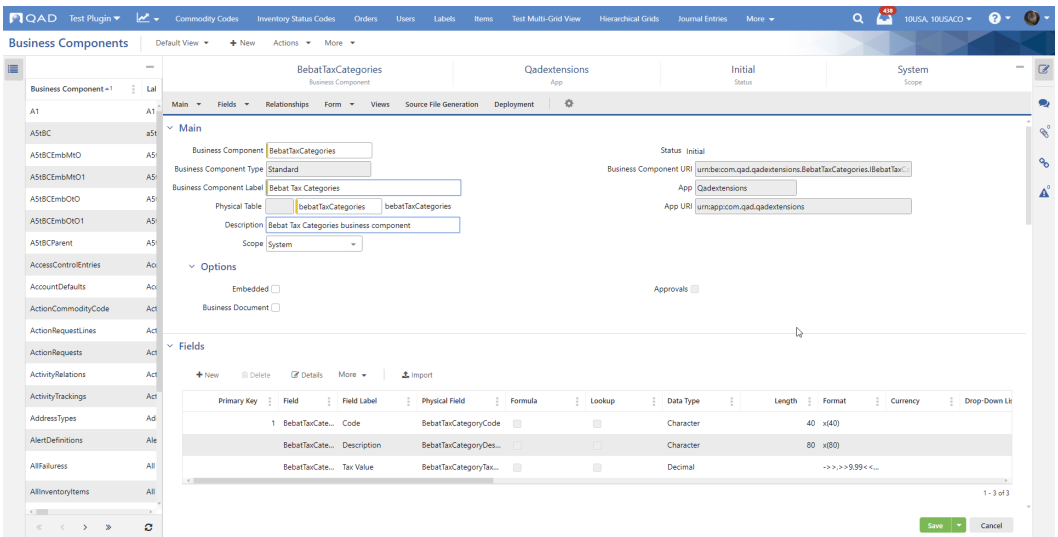
Create BebatTaxCategories Business Component

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 database: 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**.

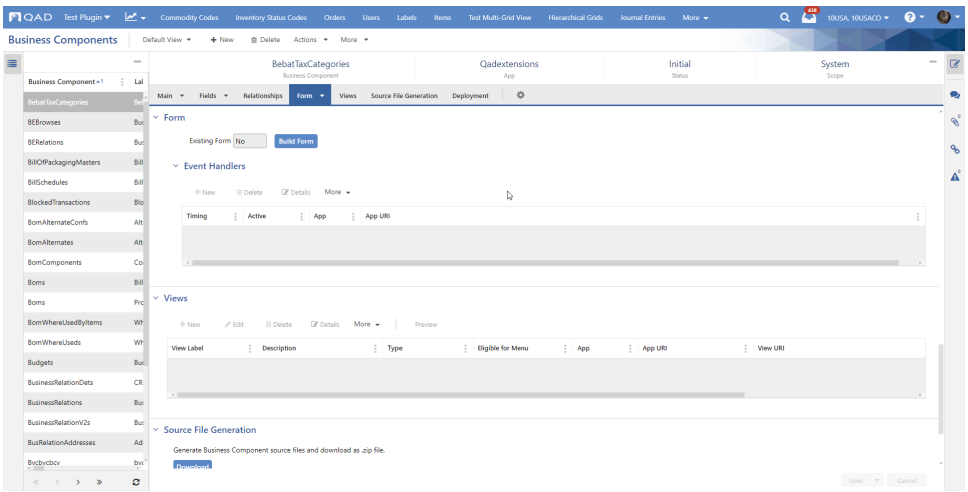
This is what the BC should look like:



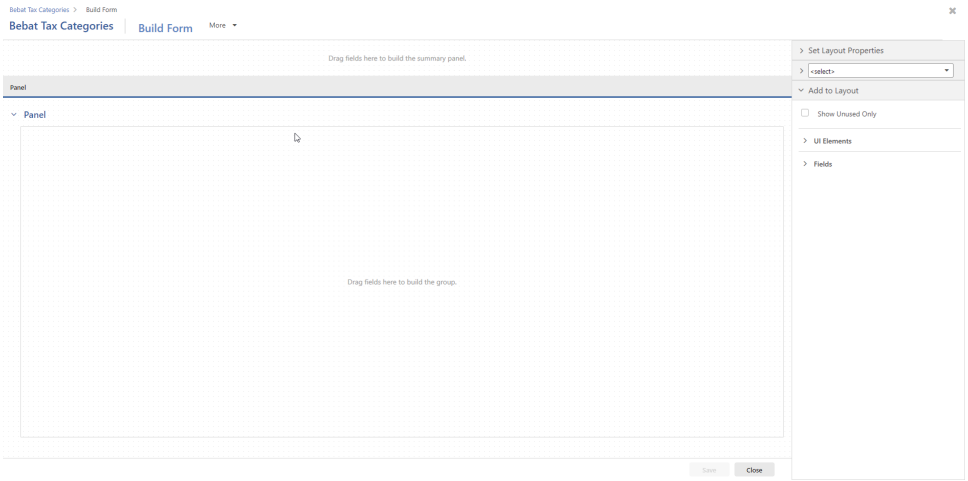
Create BebatTaxCategory View

To create a view form for the BebatTaxCategories BC, do the following steps:

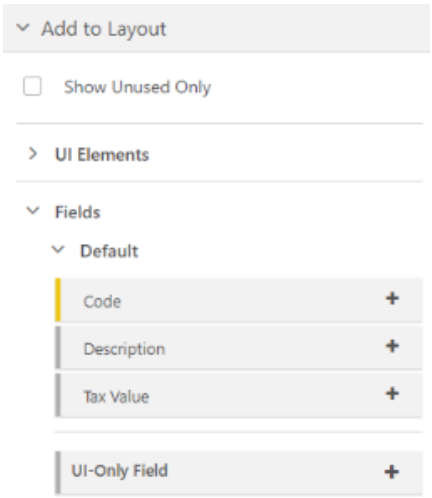
1. Open the **BebatTaxCategories** BC.
2. Open the **Form** panel.



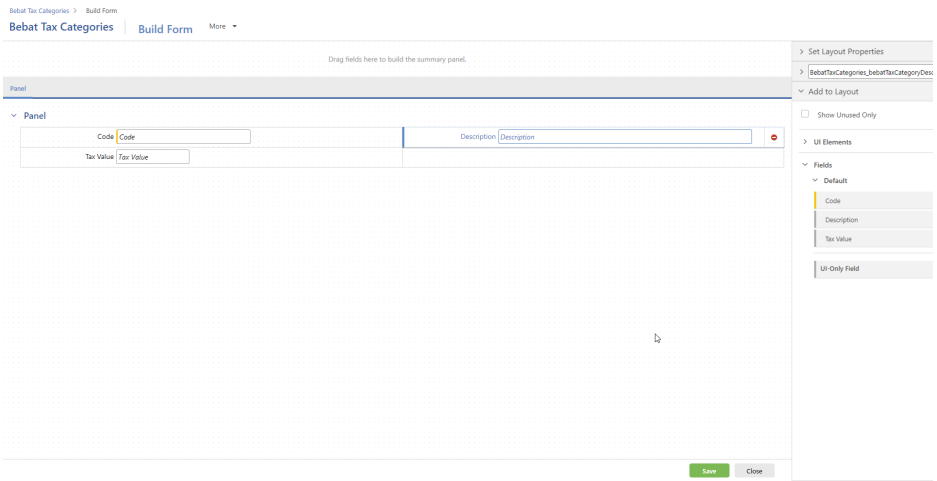
3. Click the **Build Form** button. This will open the form builder.



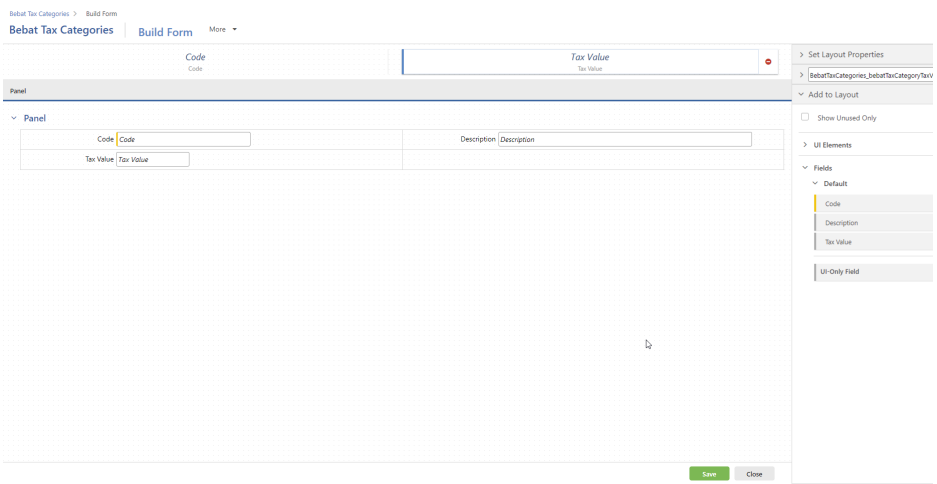
4. We now have a view with a panel that we can start dragging-and-dropping UI elements to. Follow these steps to add the elements:
    a. Click the **Fields** and **Default** menu items at the right bottom to open them.



b. Drag and drop the **Code** field on the panel.
c. Do the same for the **Description** and **Tax Value** fields so that the screen looks like this.



d. 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.

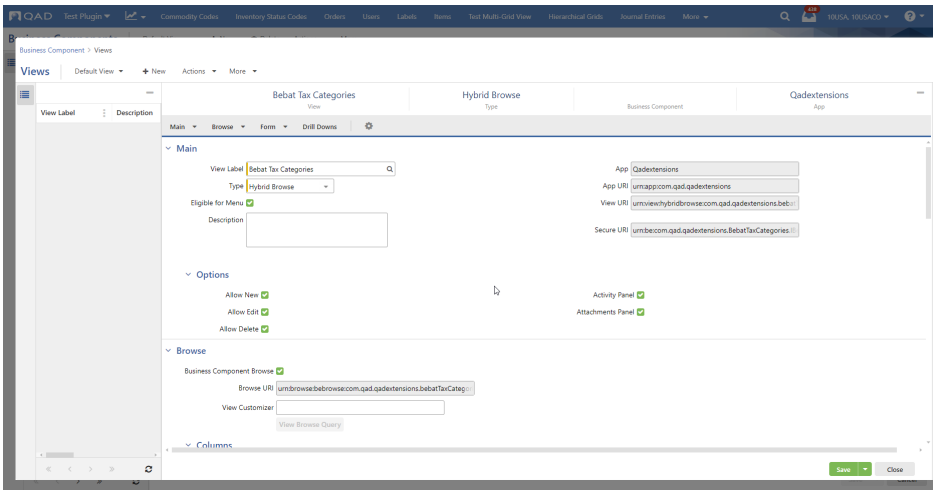


5. Now our form is ready. Click the **Save** button to save the form, and then the **Close** button.
6. After saving the form, we are back on the BC screen ready to create a Hybrid view.

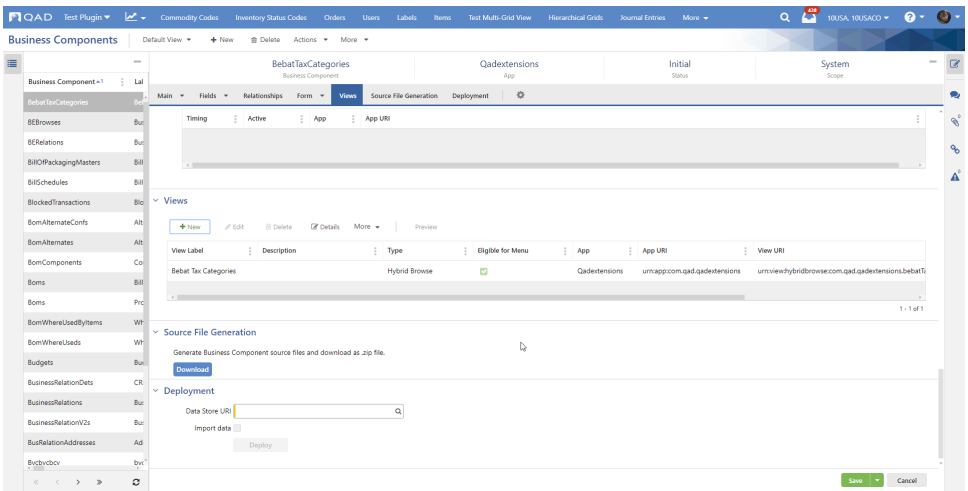
Create BebatTaxCategory Browse

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, open 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.
    - 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.

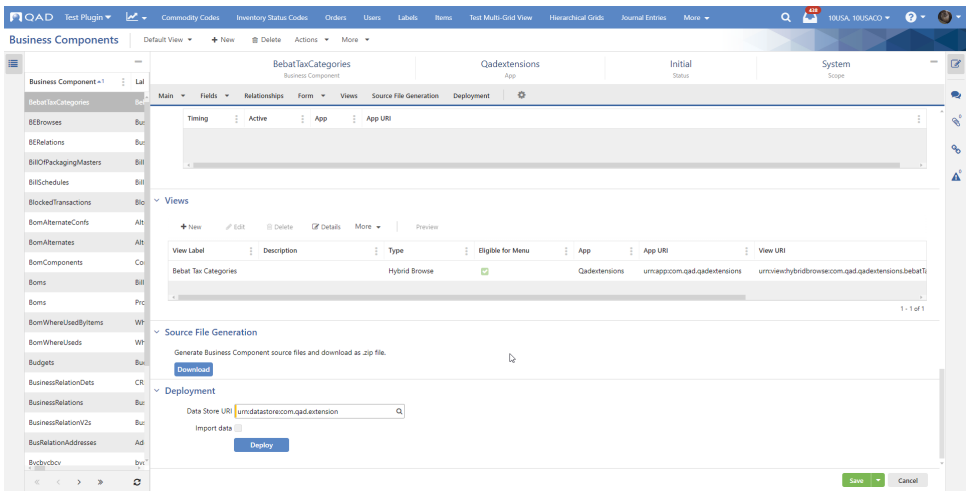


4. Click the **Save** button again to save changes.

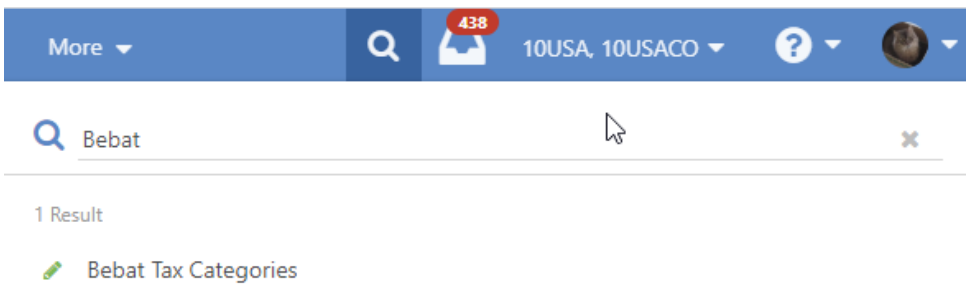
Deploy BebatTaxCategories Business Component

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:

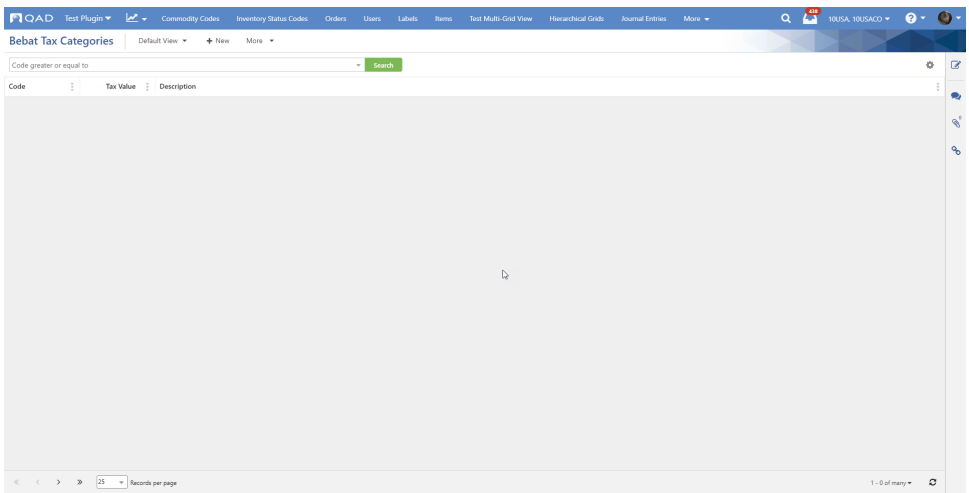
1. Open the Business Components Maintenance screen, search for the **BebatTaxCategories** business component, and then click the **Edit** button.
2. Open the **Deployment** panel and then click the lookup button on the **DataStore URI** field. This will open a lookup with the available data stores. Select 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 an embedded business component that is related to Recupel and Bebat Tax components:

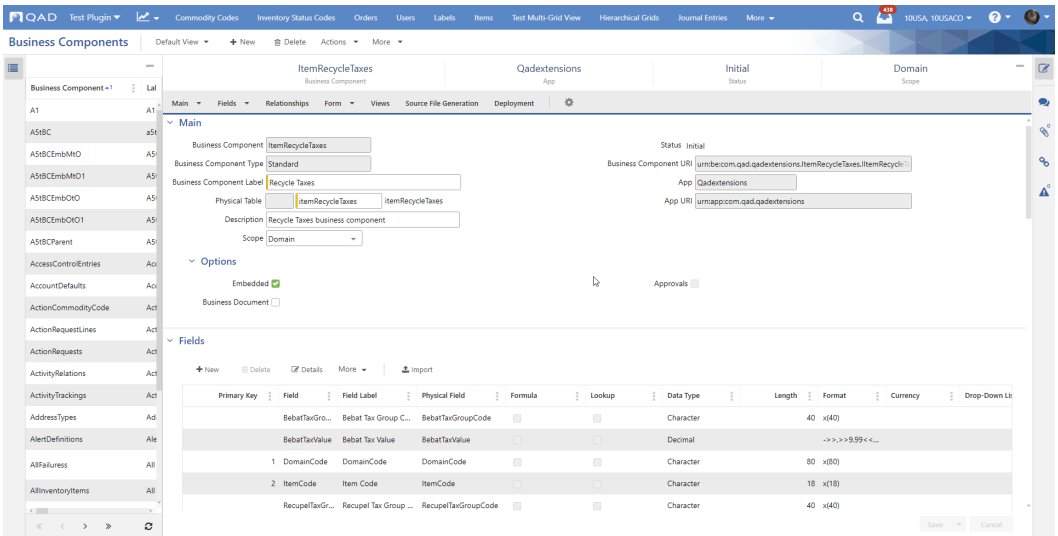
- Create ItemRecycleTaxes Embedded Business Component
- Create ItemRecycleTaxes lookup relationships
- Create ItemRecycleTaxes One-to-one relationship
- Add formula to calculate total price
- Deploy ItemRecycleTaxes Business Component

Create ItemRecycleTaxes 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 BebatTaxCategory 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 80, 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**.

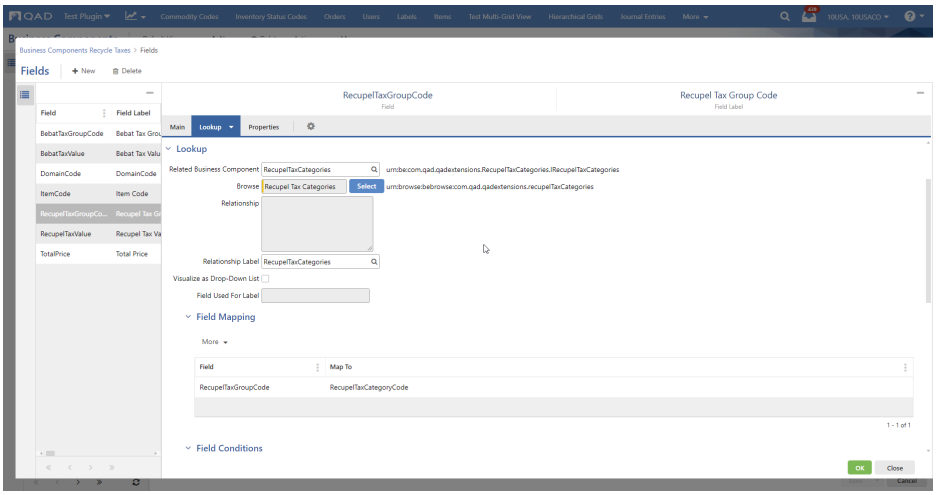
This is what the Business Component should look like:



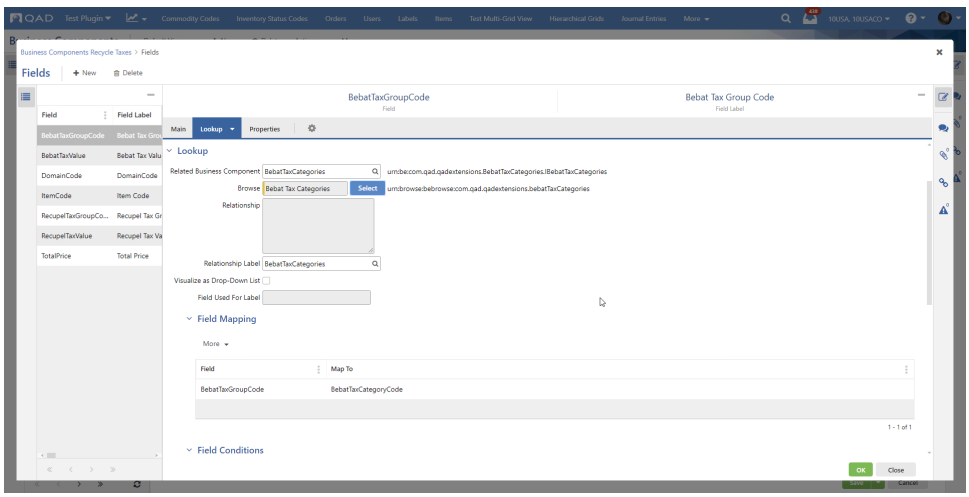
Create ItemRecycleTaxes lookup relationships

The ItemRecycleTaxes embedded BC displays Recupel and Bebat group code. For that reason, we need to add a 1-N relationship to these two 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 relationships:

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 pop-up screen appears. On this screen, enter the following values:
    - Lookup: Select the checkbox
    - Related Business Component: click the lookup icon and select the **RecupelTaxCategories** BC
    - Browse: Select the **Recupel Tax Categories** BC



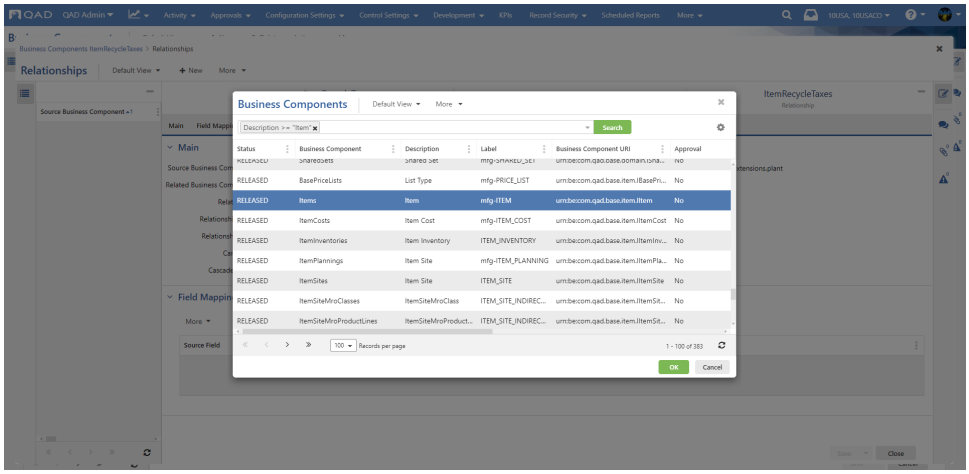
3. Click **OK**, and then click **Close**. This will bring you back to the Business Components screen.
4. We now have a relationship to Recupel Taxes, but we need a second one for Bebat Tax Group Code. Follow the same steps as for adding Recupel Taxes relationship, but select BebatTaxCategories BC instead.



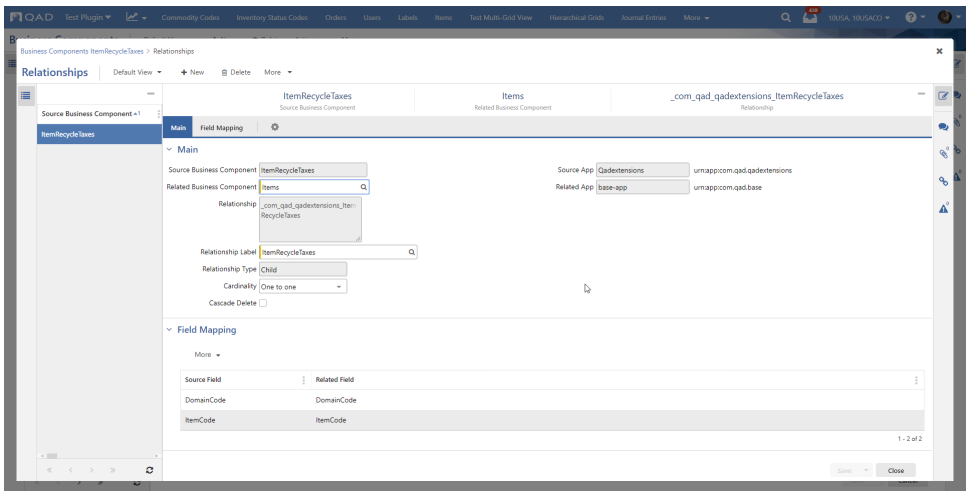
Create ItemRecycleTaxes One-to-one relationship

In order to extend an item with this BC, we need to add a One-to-one relationship to the item. Follow these steps to do that:

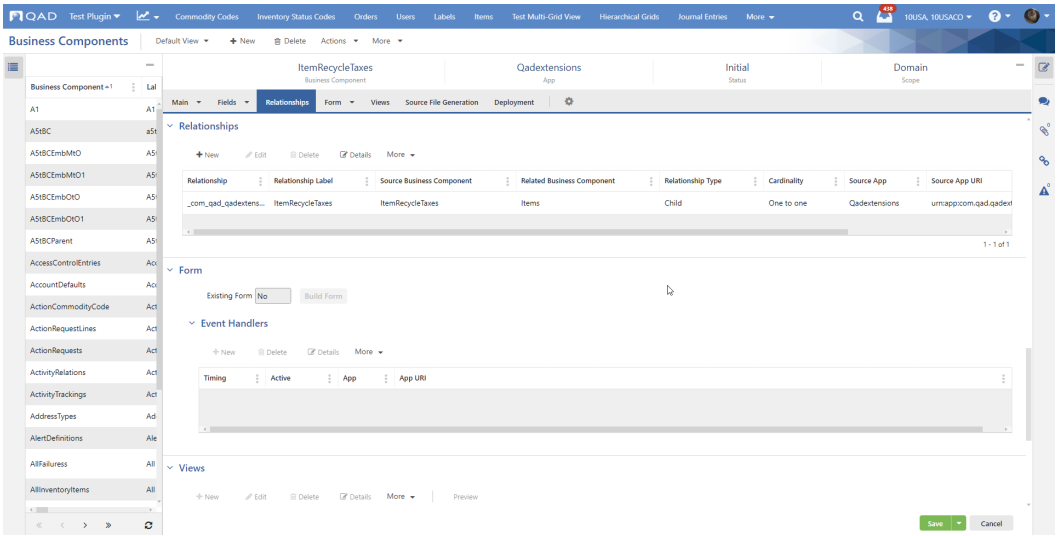
1. In **Business Components**, select **ItemRecycleTaxes**, and then click **Edit**.
2. Go to the **Relationships** panel and click **New**.
3. Click the Related Business Component lookup and select the **Items** business component. The rest of the fields automatically defaults to the correct value.



4. Click **Save** and **Close**.



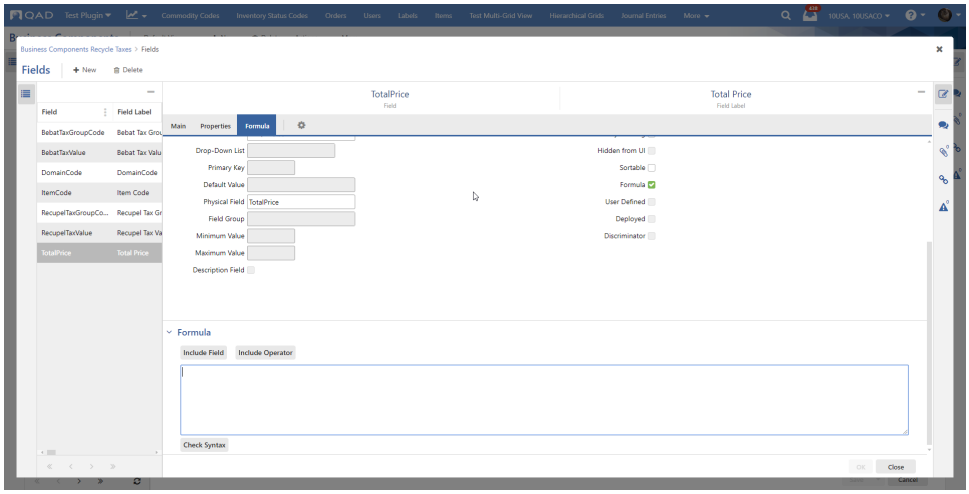
After that, your BC screen should look like this:


QAD
Test Plugin
Commodity Codes
Inventory Status Codes
Orders
Users
Labels
Items
Test Multi-Grid View
Hierarchical Grids
Journal Entries
More
10USA, 10USACD
Business Components
Default View
New
Delete
Actions
More
ItemRecycleTaxes
Business Component
Qadextensions
App
Initial
Status
Domain
Scope
Main
Fields
Relationships
Form
Views
Source File Generation
Deployment
Relationships
New
Edit
Delete
Details
More
Relationship
Relationship Label
Source Business Component
Related Business Component
Relationship Type
Cardinality
Source App
Source App URI
_com_qad_qadextens...
ItemRecycleTaxes
ItemRecycleTaxes
Items
Child
One to one
Qadextensions
1 - 1 of 1
Form
Existing Form
No
Build Form
Event Handlers
Timing
Active
App
App URI
Views
Preview
Save
Cancel

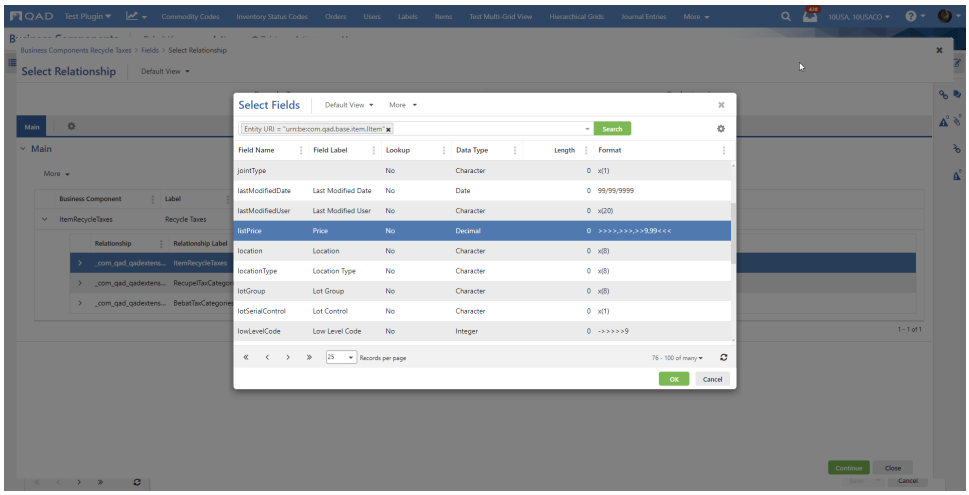
Add formula to calculate total price

On our embedded BC, 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.

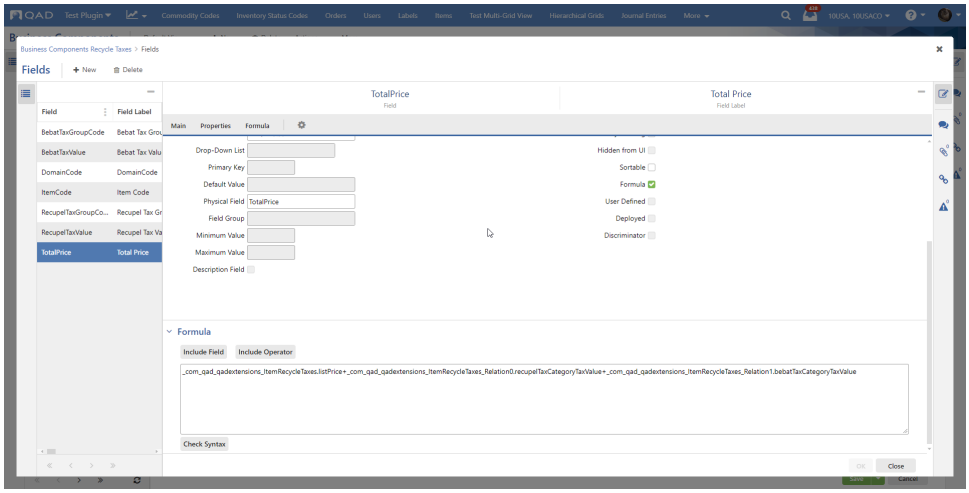
1. Open **Business Components**, select the **ItemRecycleTaxes** BC, and then click **Edit**.
2. Go to the **Fields** panel and select the **TotalPrice** field.
3. Select the **Formula** field checkbox.
4. Click the **Details** button, and then open the **Formula** panel.



5. Click the **Include Field** button and from the Items relationship, select the **listPrice** field.



6. Then, enter + and click the **Include Field** button again and select the RecupelTaxValue field from the corresponding relationship; do the same with BebatTaxValue so that you end up with this formula: see screen shot below.
7. Click the **Check Syntax** button.

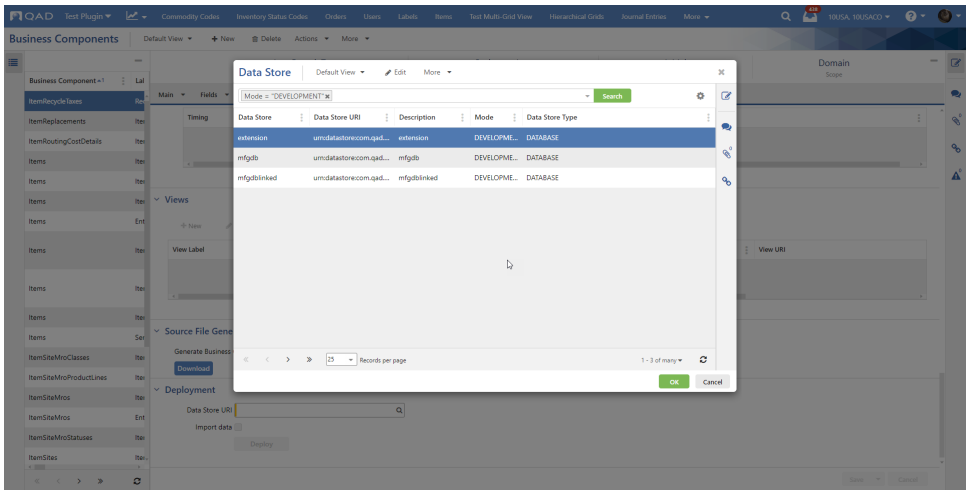


8. Click **OK** and **Close**.

Deploy ItemRecycleTaxes Business Component

The last step to make this embedded BC active is to deploy it. Follow these steps:

1. Open **Business Components**, search for the **ItemRecycleTaxes** BC, and then click **Edit**.
2. Open 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**.

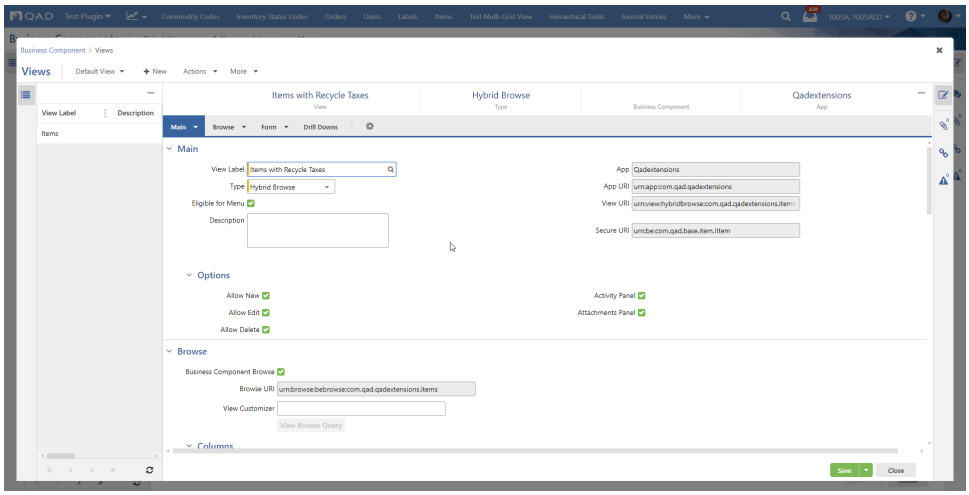


3. Click the **Deploy** button to deploy the BC and make it active.

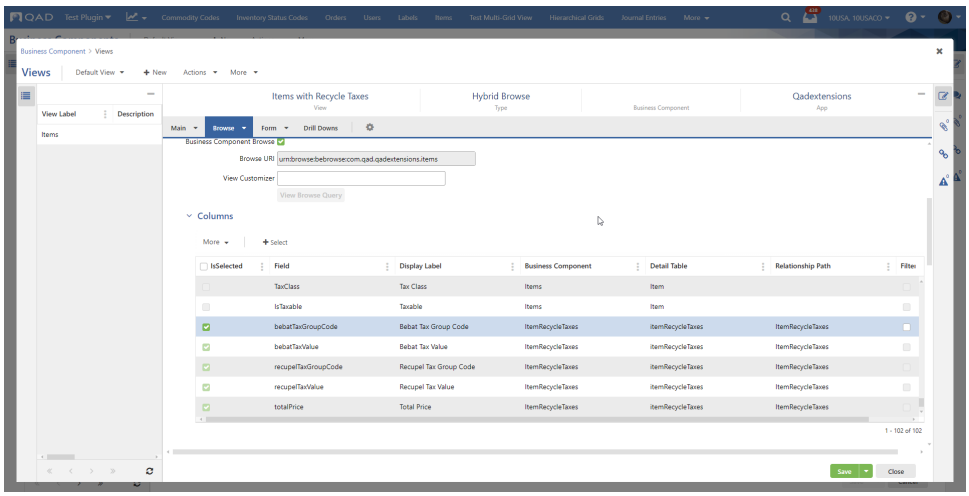
Create Items with Recycle Taxes BC browse

On this page, we show how to create a Business Component browse that contains fields from the BC itself, but also from its extension.

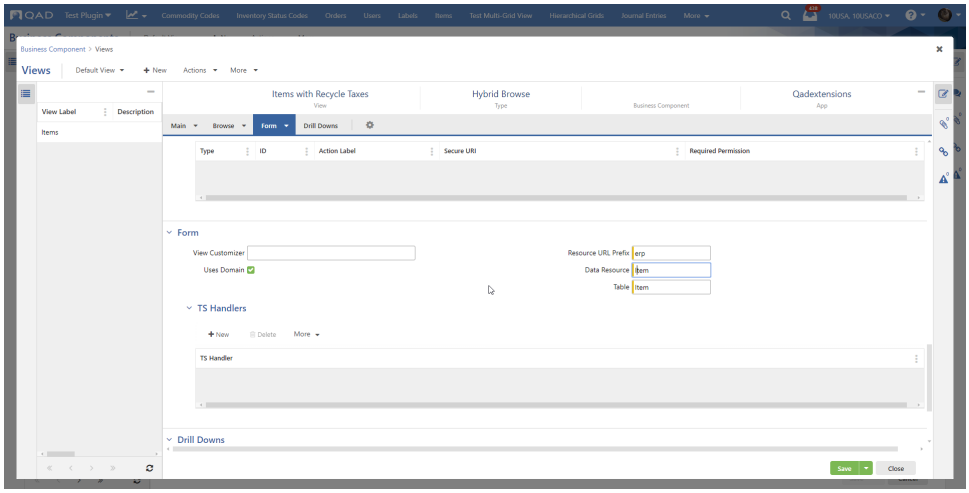
1. Open **Business Components**, search for the **Items** BC, and then click **Edit**.
2. Click the Views panel, and then click the **New** button.
3. Enter a label for the Hybrid View, this is the label you'll see on the menu.



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 grid.



5. After that, fill in the following values (Resource URL Prefix, Data Resource, and Table).

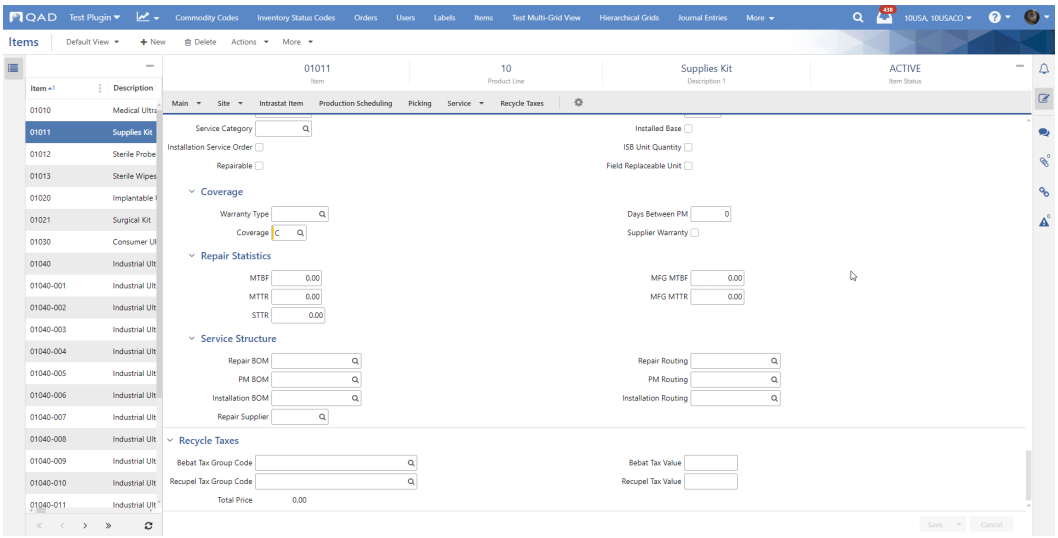


6. Click **Save** and **Close**.

That's it, now you should be able to run the browse from the menu.

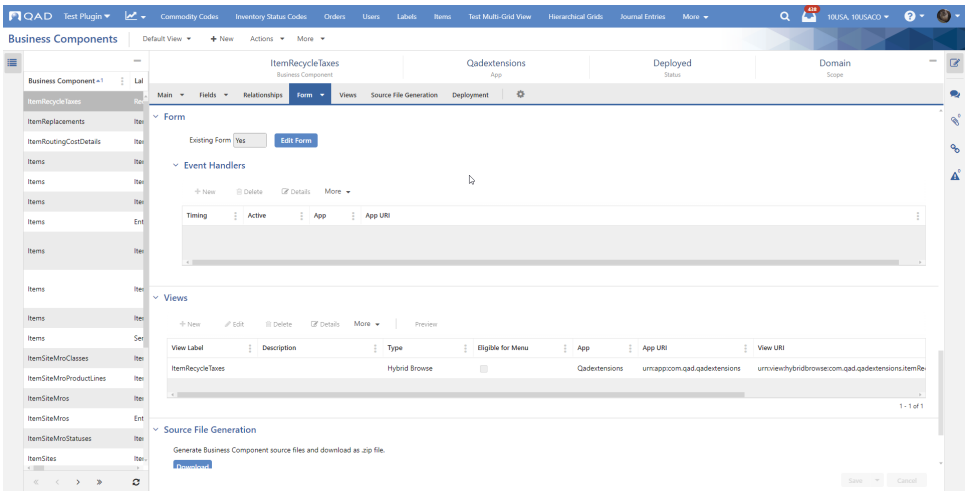
Make tax value fields read only on view

After deploying the **ItemRecycleTaxes** business component, we should see the fields from ItemRecycleTaxes on the Items view.

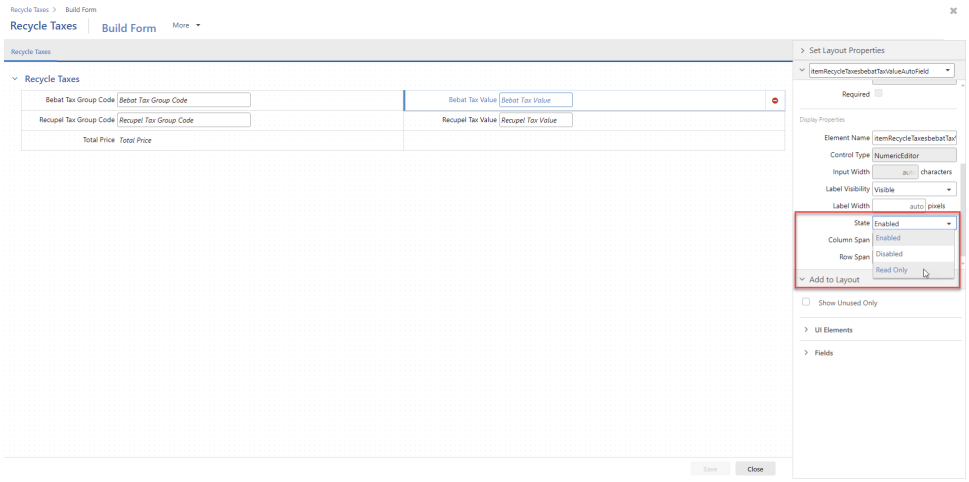


Because the Tax Value fields need to be automatically filled when a Group Code is selected, we need to make them read-only. This can be done using the form builder:

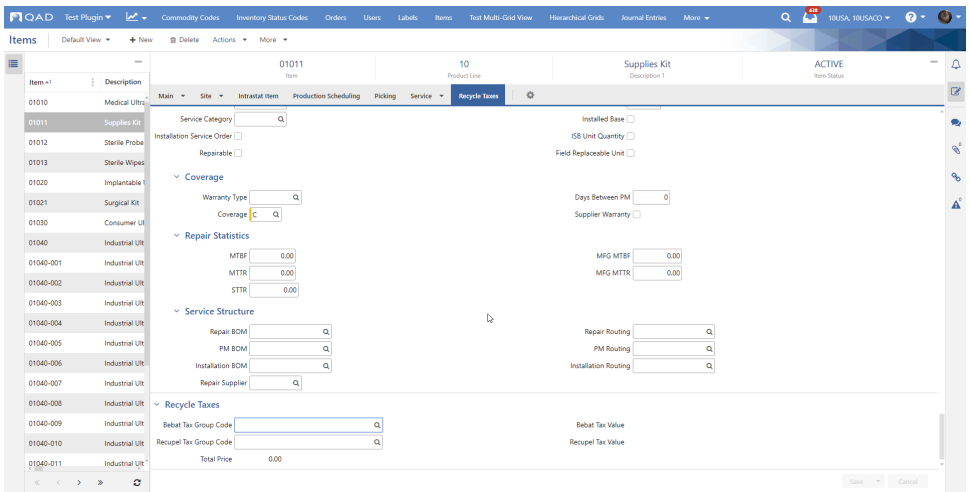
1. Open **Business Components**, search for the **ItemRecycleTaxes** BC, and then click **Edit**.
2. Click the **Form** panel, and then click the **Edit Form** button.



3. Select the **Bebat Tax Value** field and go to **Display Properties** at the right.



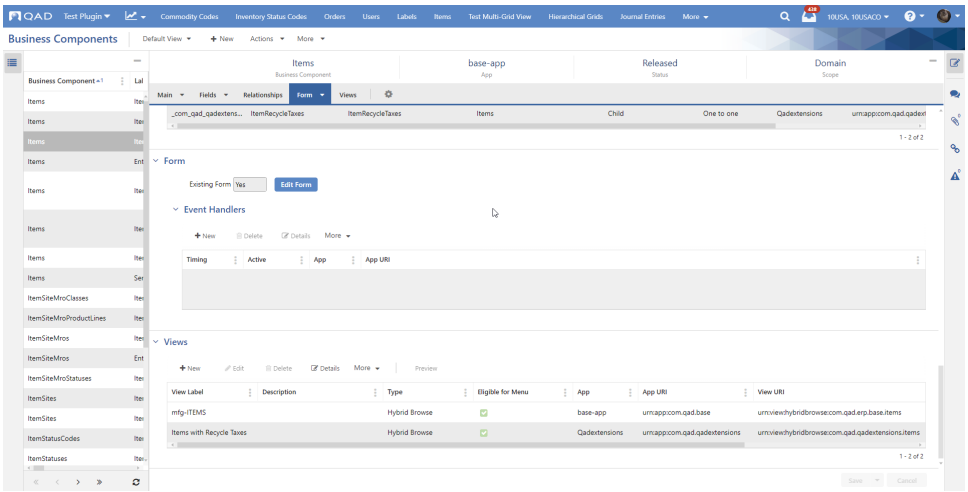
4. From the **State** drop-down menu, select **Read Only**.
5. Do the same for the **Recupel Tax Value** field.
6. Click **Save** and **Close**.
7. Now, go back to the Items screen, but be sure to click the **Refresh** button from the browses, so that it completely reloads. Open an item, and see that the two fields are read-only now.



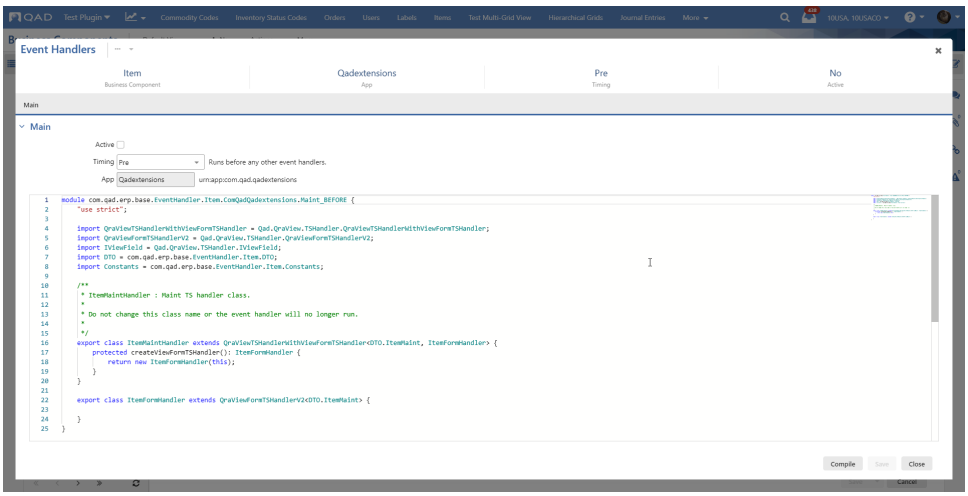
Add event handler to retrieve Tax Value data

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. Open the **Form** panel.



3. On the **Event Handlers** panel, click the **New** button.



4. 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
         *
```

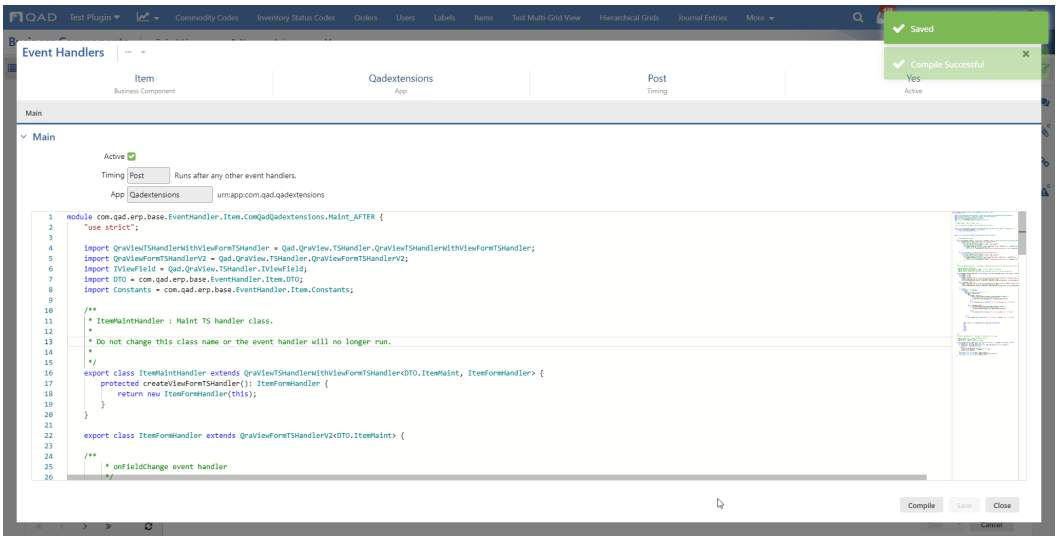
```
         * @param submitresults : array of SubmitResult objects
         * @param message : message to show
         * @param status : status to show in the message
         */
        private handleServerError(submitresults: Qad.Common.DTO.SubmitResult , message:string,
status: string) {
            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;
                    });
                }
            }
        }
```

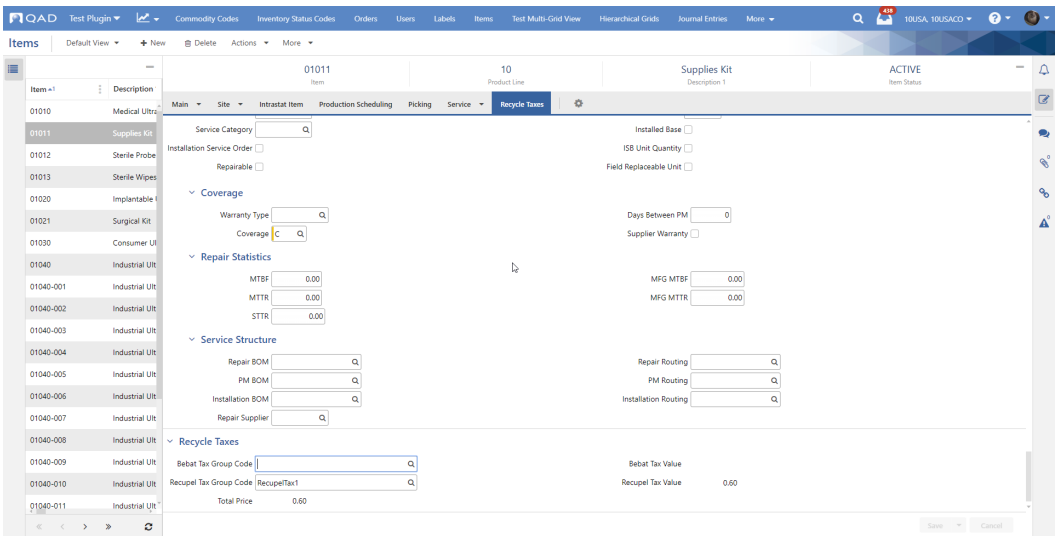
As you can see above, we implement the onFieldChange event by simply overriding the base class method. Inside the method, we check the field name and call the getRecycleTaxValue method to retrieve the tax value. On return of that value, we get a reference to the right tax value field and update its value.

4. Set the event handler **Active**, click the **Compile** button and **Save**:



5. After this we are ready to test: open the Items view, select a record, and then 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 OOABL validation to check if a tax category is still referenced when deleting (using IVirtualBusinessEntity)

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. As we need to extend the data extension which was created for the tax categories, this OOABL extension will be done on the **IVirtualBusinessEntity.**

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.extensions.hackathon.HackathonMessageKey.

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 available (ItemRecycleTax)
                then do:
                    assign msg = new Message(HackathonMessageKey:ITEM_EXISTS_WITH_RECUPEL, 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.ComQadQadextensions.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") {
```

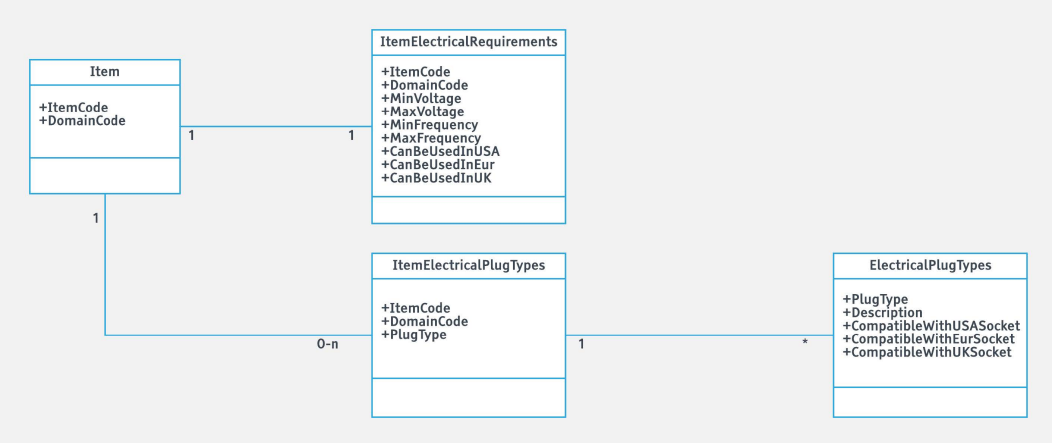
```
                targetUrl="api/qracore/be/urn:be:com.extensions.qadextensionsP.
RecupelTaxCategories.IRecupelTaxCategories?domainCode={domainCode}&recupelTaxCategoryCode=
{recupelTaxCategoryCode}";
                //replace placeholders in the url with the domain code and tax code
                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 Many-to-one relationship

We will add the information to the item about its electrical compatibility. 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 One-to-one relationship between Item and ItemElectricalRequirements is a child-parent relationship (because we want to add extra fields on the items UI). Also, the Many-to-one relationship between ItemElectrical PlugTypes and Item is a child-parent relationship (to add a grid to the items UI).

The 1-N relationships to ElectricalPlugTypes is a lookup relationship. 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 Relationship for more info on the different types of relationships.

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 the Electrical Plug Types business component:

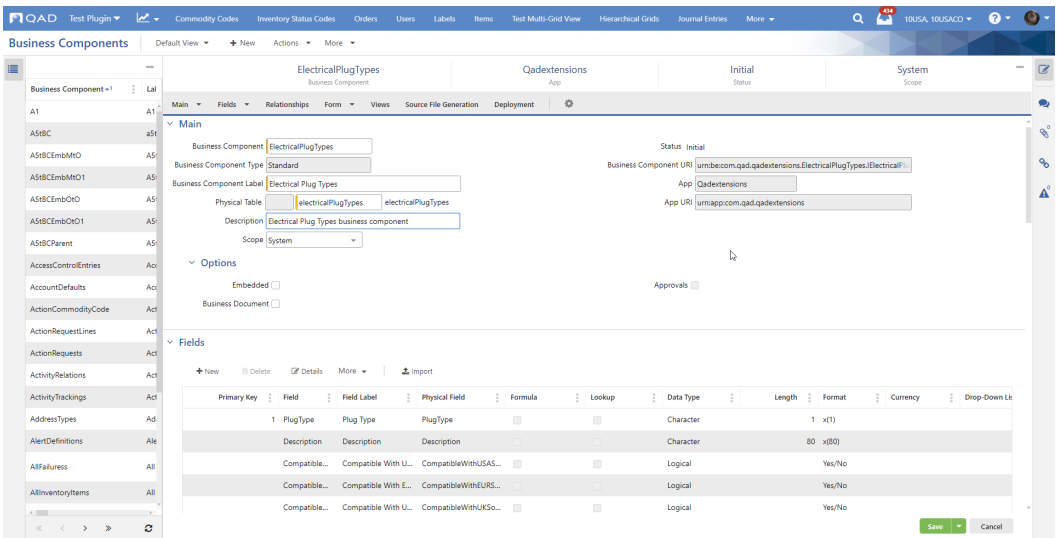
Create ElectricalPlugTypes Business Component

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 database: 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 (Primary Key 1, 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 (Label Description, Type Character, length 80)
        - 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**.

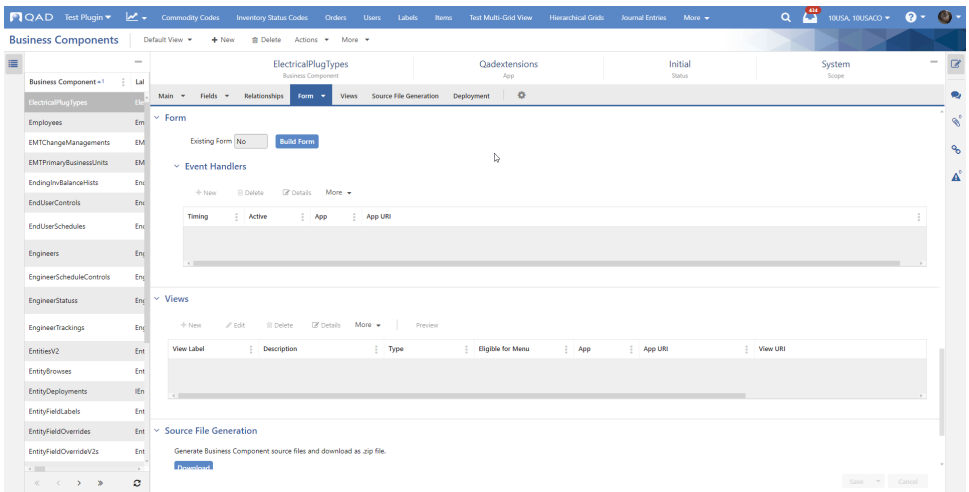
This is what the BC should look like:



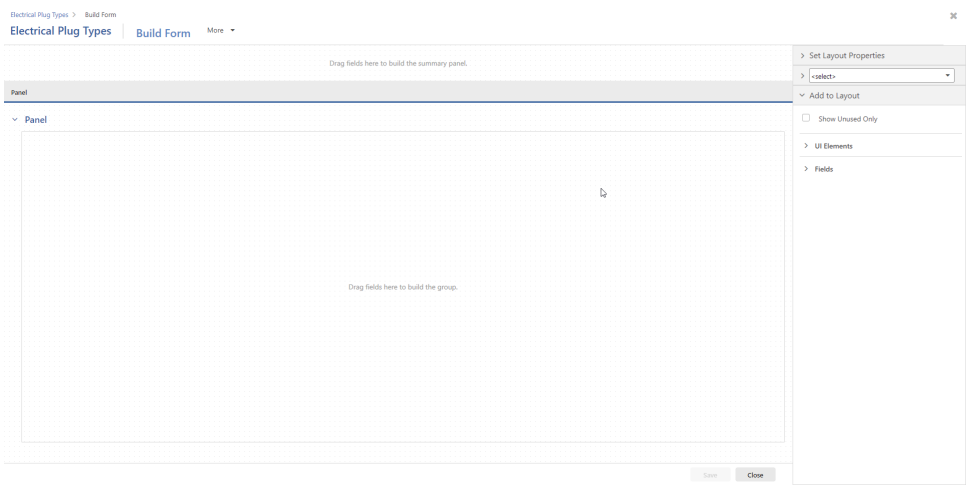
Create ElectricalPlugTypes View

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:

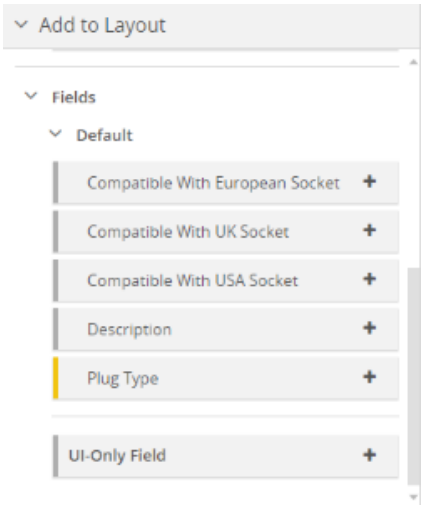
1. Open the **ElectricalPlugTypes** BC.
2. Click **Form** to go to the **Form** panel.



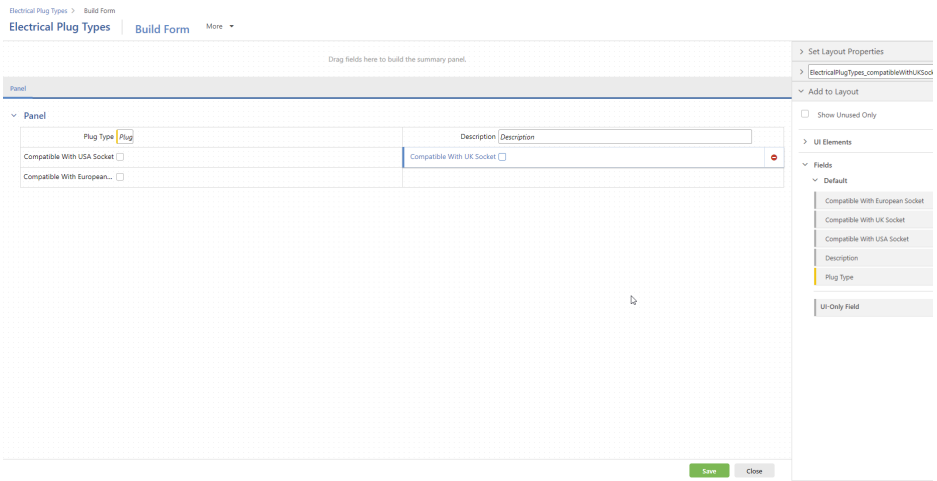
3. Click the **Build Form** button. This will open the form builder.



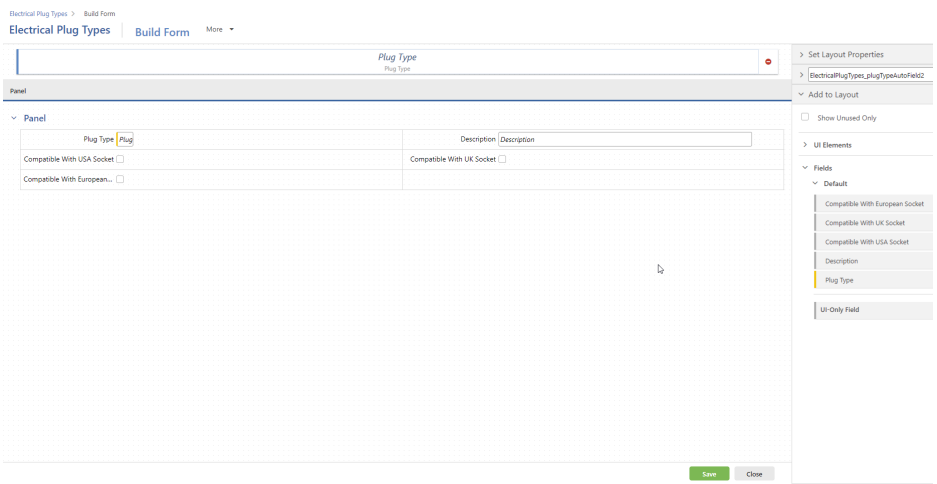
4. We now have a view with a panel that we can start dragging-and-dropping UI elements to. Follow these steps to add the elements:
    a. Click the **Fields** and **Default** menu items at the right bottom to open them.



b. Now select the **Plug Type** field and drop it on the panel.
c. Do the same for **Description** and the other fields so that the screen looks like this.



d. The next step is to create the **Summary** panel. This can be done by dropping **Plug Type** on the **Summary** panel at the top of the screen.



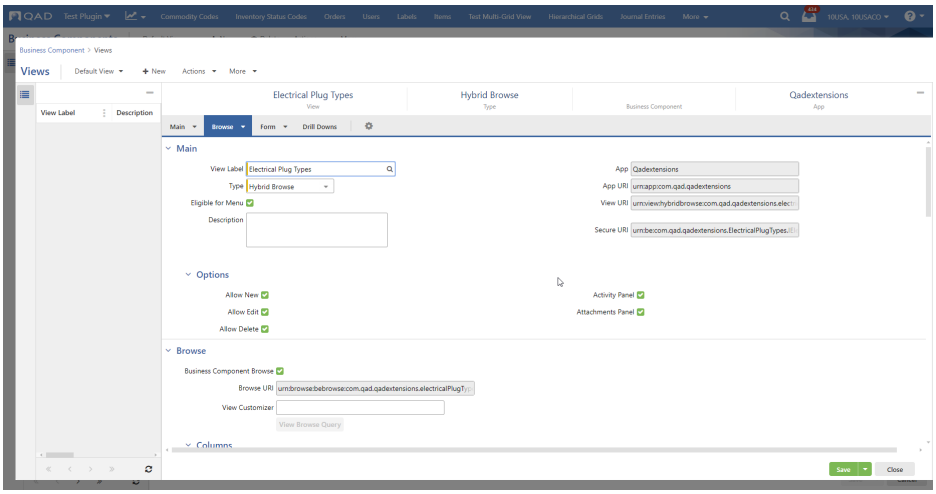
5. Now our view form is ready. Click **Save**, and then **Close**.
6. After saving the view form, we are back on the BC screen ready to create a Hybrid view.

Create ElectricalPlugTypes Browse

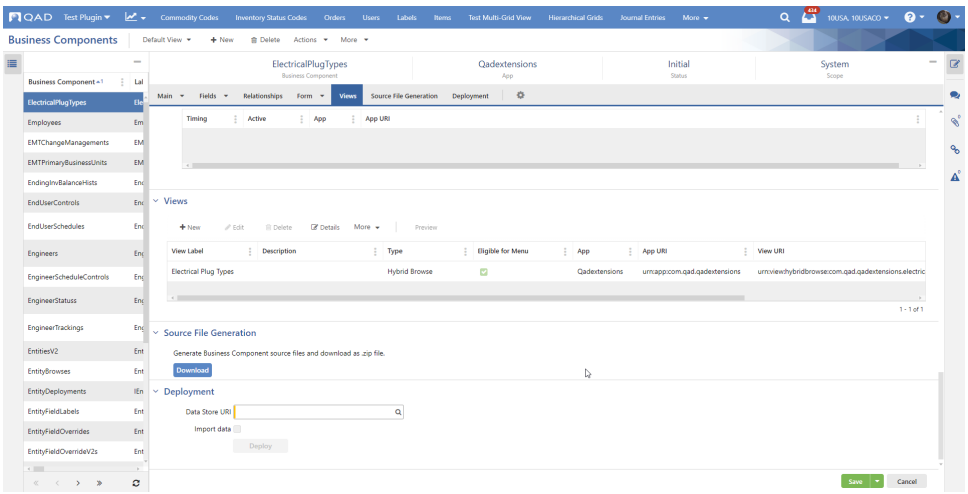
In order to see the ElectricalPlugTypes BC in the menu and open a browse on it, we need to create a BC browse. This can be done by following these steps:

1. On the ElectricalPlugTypes screen, open 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, clear the **Description** column. This column is too big to show in a browse, this way we remove it from the browse view.



2. Click **Save** to save this browse, and then **Close**.
3. After saving the new Hybrid view, the UI looks as follows.

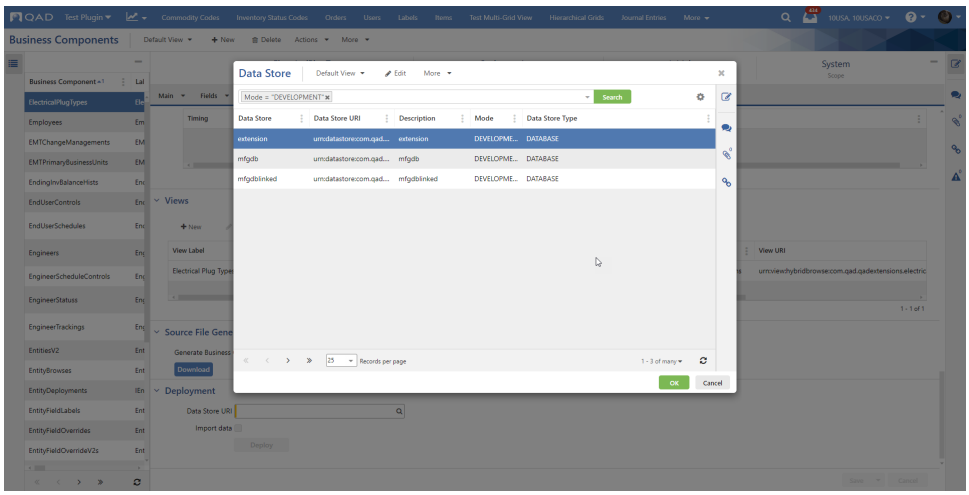


4. Now click **Save** again to save everything.

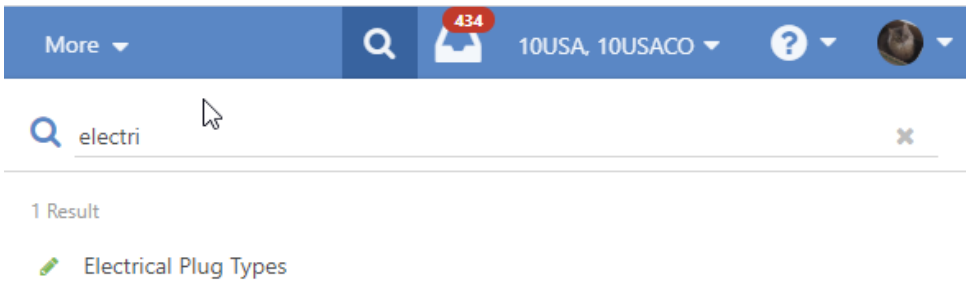
Deploy ElectricalPlugTypes Business Component

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:

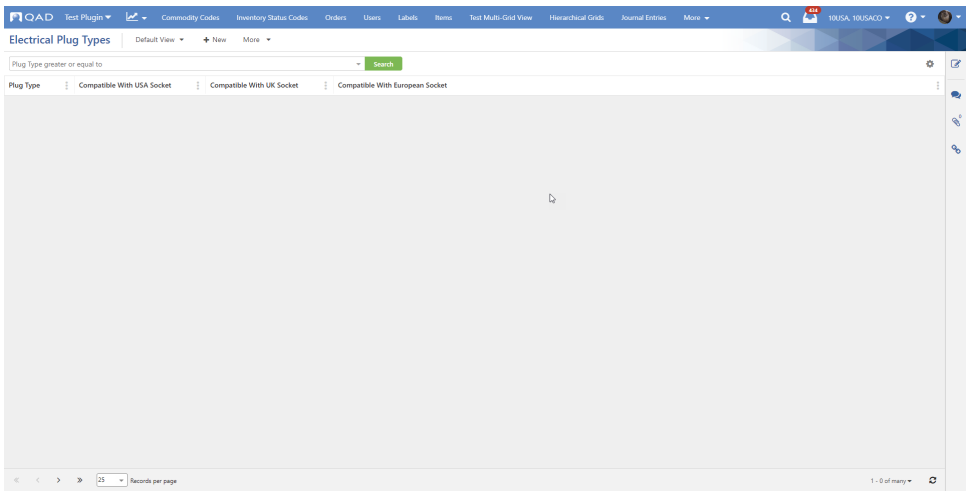
1. Open the Business Components Maintenance screen, search for the **ElectricalPlugTypes** business component, and then click the **Edit** button.
2. Scroll to the bottom of the screen and click the lookup button on the **DataStore URI** field. This will open a lookup with the available data stores. 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

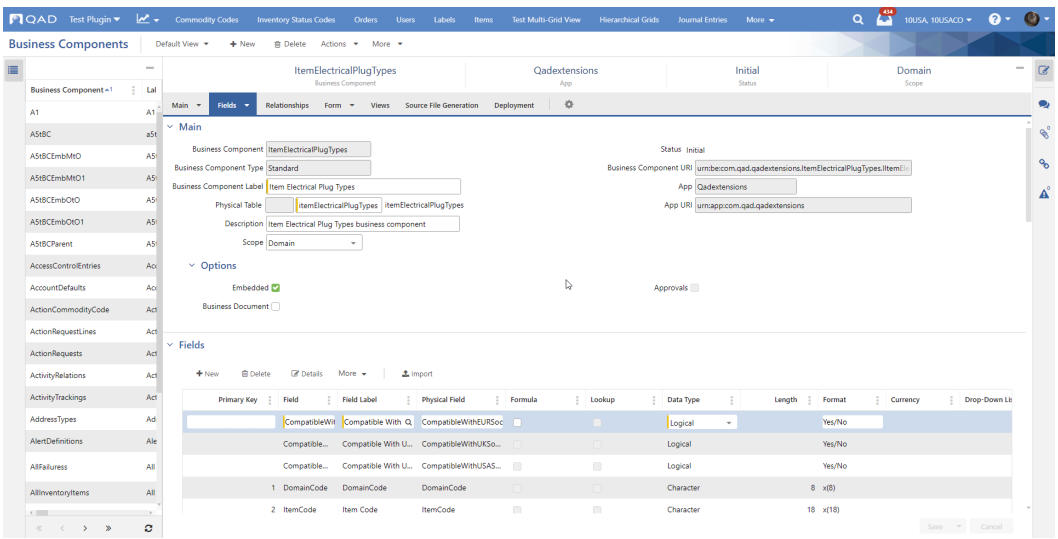
This section explains all the steps to create an embedded BC that is related to the Items component:

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 embedded BC 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)
        - PlugID: Plug id (label Plug ID, type Character, length 1, format x(1), key field 3)
        - 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**.

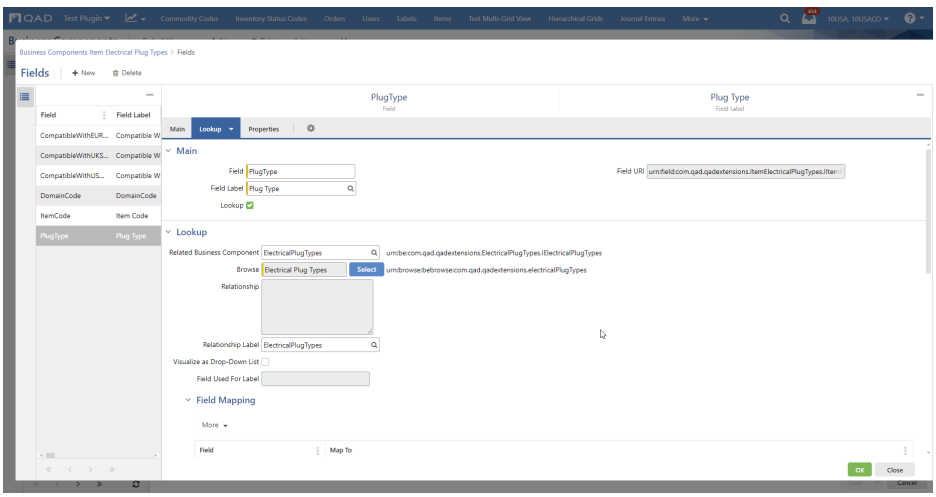
This is how the Business Component should look like:



Create ItemElectricalPlugTypes lookup relationships

The ItemElectricalPlugTypes embedded BC displays Electrical Plug Types. For that reason, we need to add a 1-N relationship 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 relationship:

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 pop-up screen appears. On this screen, enter the following values:
    - Lookup: Select the checkbox
    - Related Business Component: click the lookup icon and select the **Electrical Plug Types** BC
    - Browse: Select the **Electrical Plug Types** BC

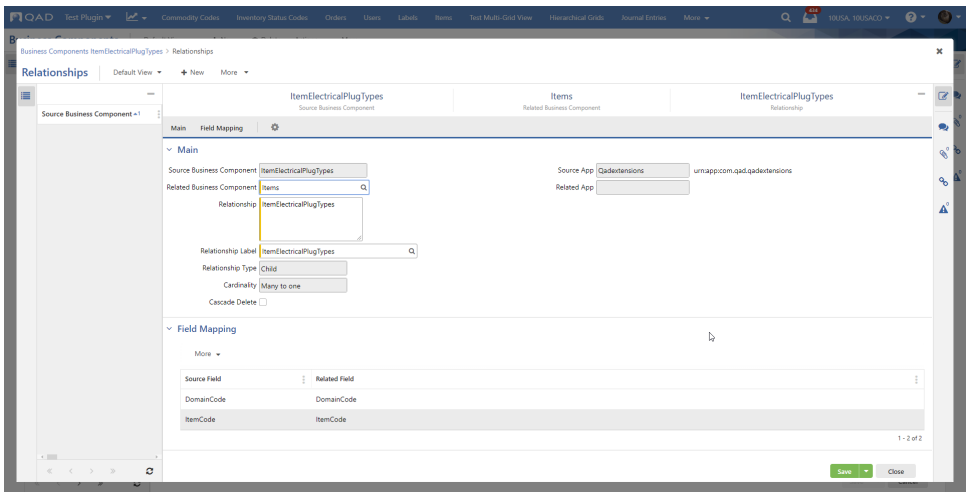


3. Click **OK**, and then click **Close**. This will bring you back to the Business Components screen.
4. Click the **Save** button again to finish this task.

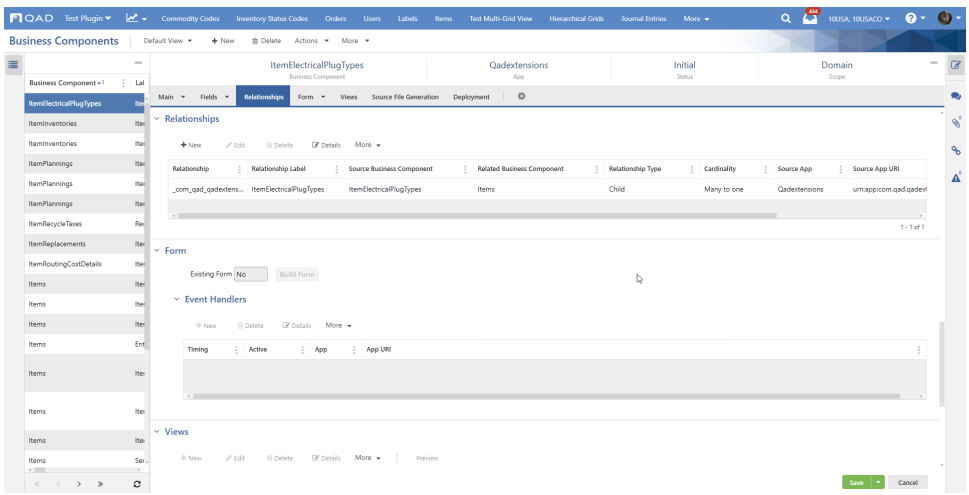
Create ItemElectricalPlugTypes Many-to-one relationship

In order to extend Items with this BC, we need to add a Many-to-one relationship to the Items. 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:

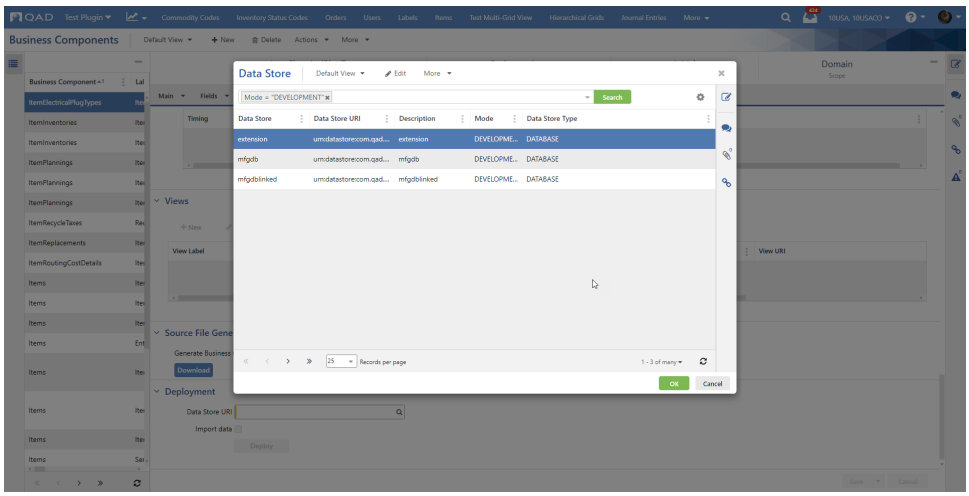


7. Click **Save** again to finish this task.

Deploy ItemElectricalPlugTypes Business Component

The last step to make this embedded BC active is to deploy it. Follow these steps to do so:

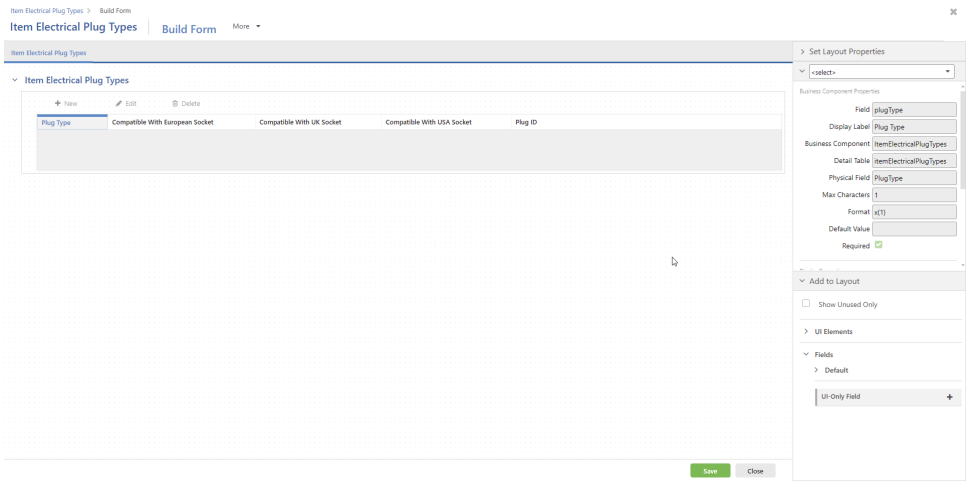
1. Open **Business Components**, select the **Item Electrical Plug Types** component, and then click **Edit**.
2. Go to the bottom of the screen, click the **Data Store URI** lookup, and then select your development database:



3. Click **OK**.
4. Click the **Deploy** button to deploy.
5. Now when opening the Items 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** business 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 Items component:

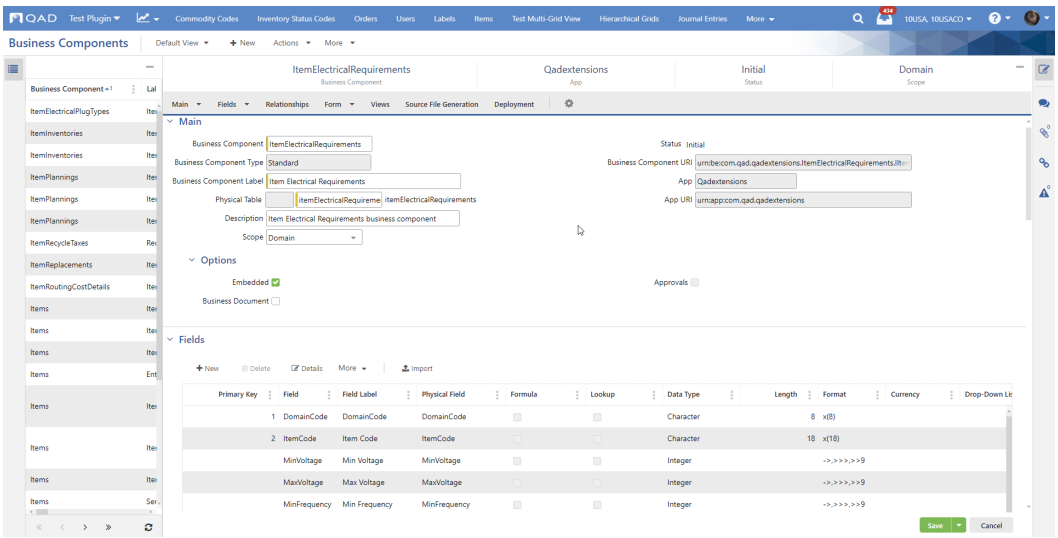
- Create ItemElectricalRequirements Embedded Business Component
- Create ItemElectricalRequirements One-to-one relationship
- Deploy ItemElectricalRequirements Business Component

Create ItemElectricalRequirements Embedded Business Component

In order to extend Items maintenance with fields for the electrical requirements info, we create the ItemElectricalRequirements embedded BC. This BC will bring ItemElectricalRequirements together with Items. 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 embedded BC 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, type Integer)
        - MaxVoltage: This is the maximum voltage the item can work with (label Max Voltage, type Integer)
        - MinFrequency: This is the minimum frequency the item can work with (label Min Frequency, type Integer)
        - MaxFrequency: This is the maximum frequency the item can work with (label Max Frequency, type 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**.

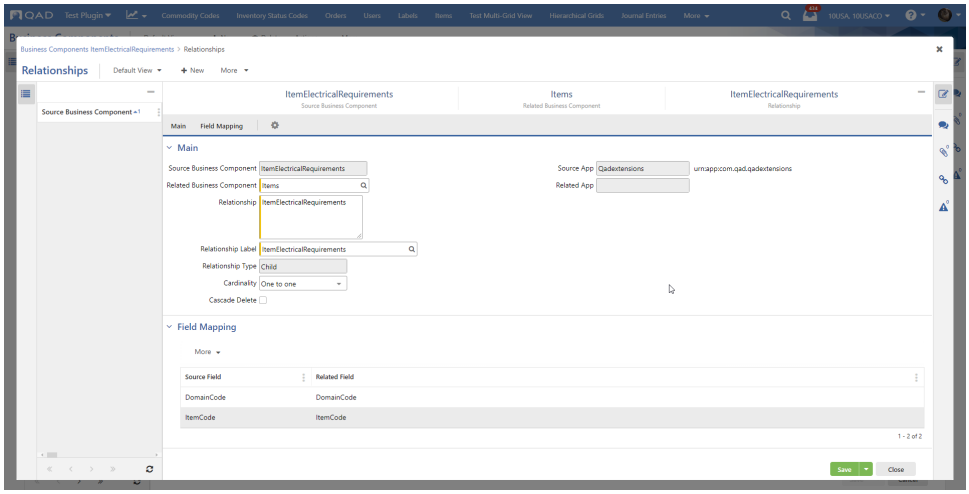
This is how the Business Component should look like:



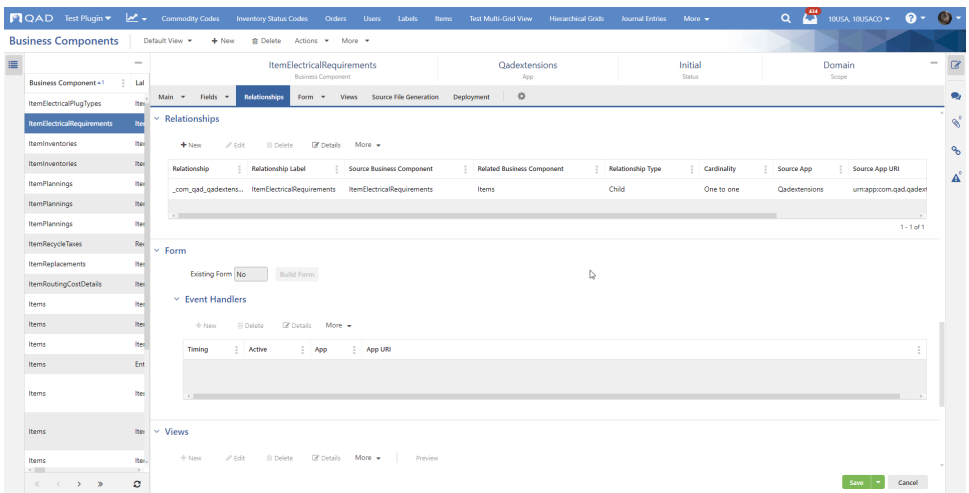
Create ItemElectricalRequirements One-to-one relationship

In order to extend Items with this BC, we need to add a One-to-one relationship to the Items. 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:

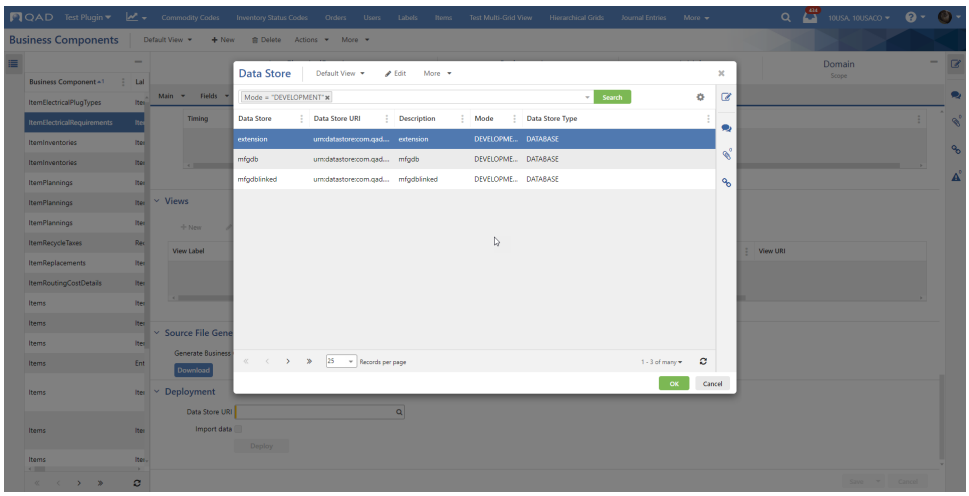


7. Click **Save** again to finish this task.

Deploy ItemElectricalRequirements Business Component

The last step to make this embedded 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 database:

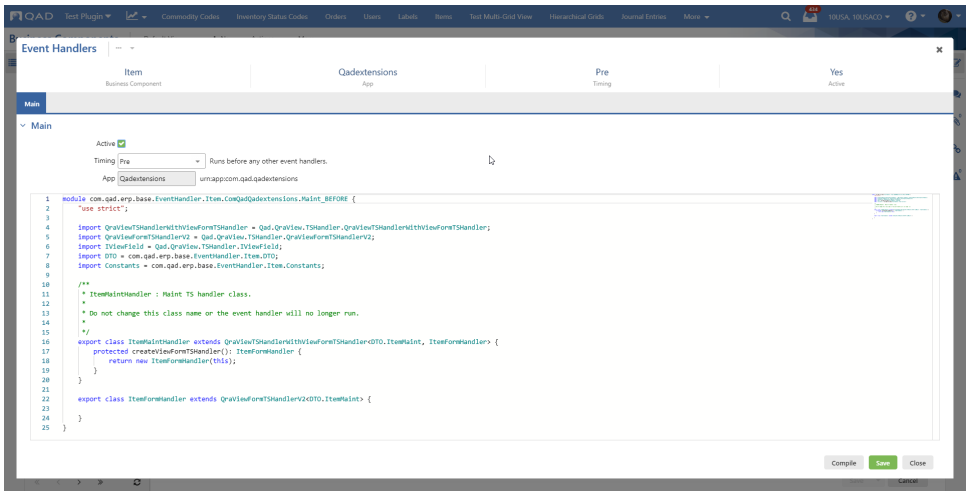


3. Click **OK**.
4. Click the **Deploy** button to deploy.
5. Now when opening the Items BC, you should see an extra panel with electrical requirements info.

Add event handler to retrieve electrical plug info

At this point, we have an 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 **Form** > **Event Handlers** and click **New** to 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 plug type
            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 **Form** > **Event Handlers** and click **New** to 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.

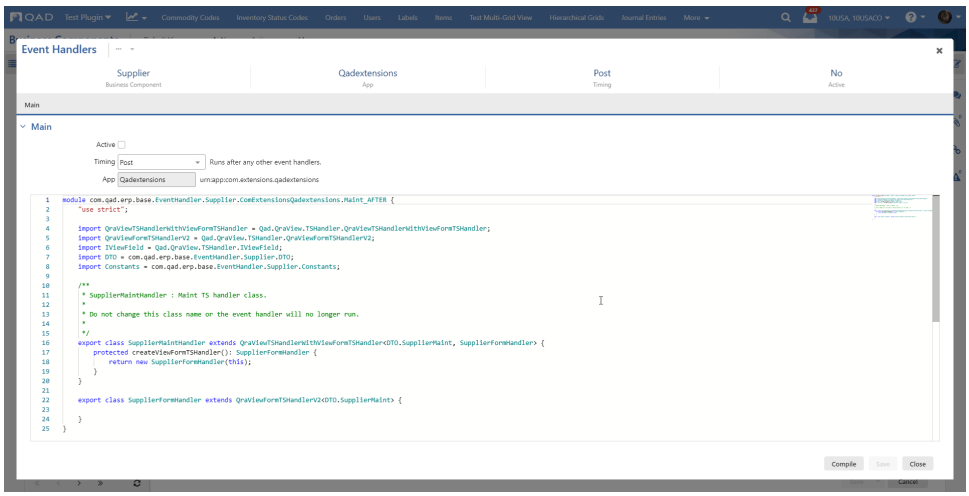
Steps to follow:

- 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 Components**, select **Suppliers** (base app), and then click **Edit**.
2. Go to **Form** > **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;


            onAfterViewInit(eventData: EventData.QraView.AfterViewInitEventData): void {
                this.addHTML();
                //create the Kendo dropdown
                (<SupplierFormHandler>this.ViewFormTSHandler).createCityCustomDropDown();
            }

        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;


            onAfterViewInit(eventData: EventData.QraView.AfterViewInitEventData): void {
                this.addHTML();
                //create the Kendo dropdown
                (<SupplierFormHandler>this.ViewFormTSHandler).createCityCustomDropDown();

            }

                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> {
        // create the kendo dropdown
        public 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 and resetUIStateForNoZippopotamData methods 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.ComExtensionsQadextensions.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;

        onAfterViewInit(eventData: EventData.QraView.AfterViewInitEventData): void {
            this.addHTML();
            //create the Kendo dropdown
            (<SupplierFormHandler>this.ViewFormTSHandler).createCityCustomDropDown();
        }

        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\" >");
        }


        protected onDestroy() {
            //remove the html we added
            this._appendedHtml1.remove();
            this._appendedHtml2.remove();
        }

        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();
        }

    }

    export class SupplierFormHandler extends QraViewFormTSHandlerV2<DTO.SupplierMaint> {
        // create the kendo dropdown
        public 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 easily 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 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();
        }

        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;
    }
}
```

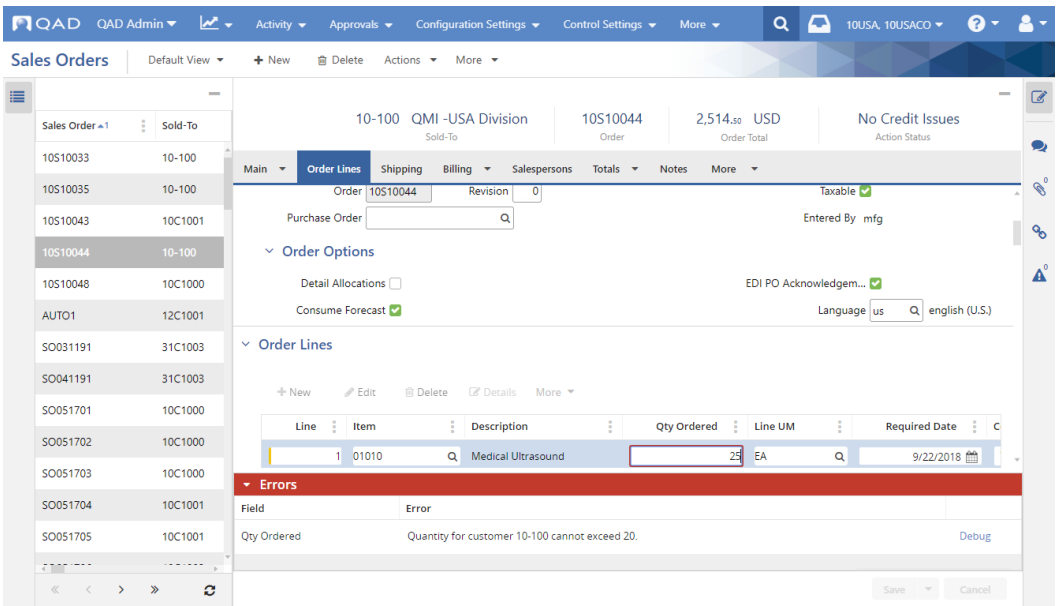
Example 3 - Extend the OOABL: Add extra validation to Sales Order

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 the 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.

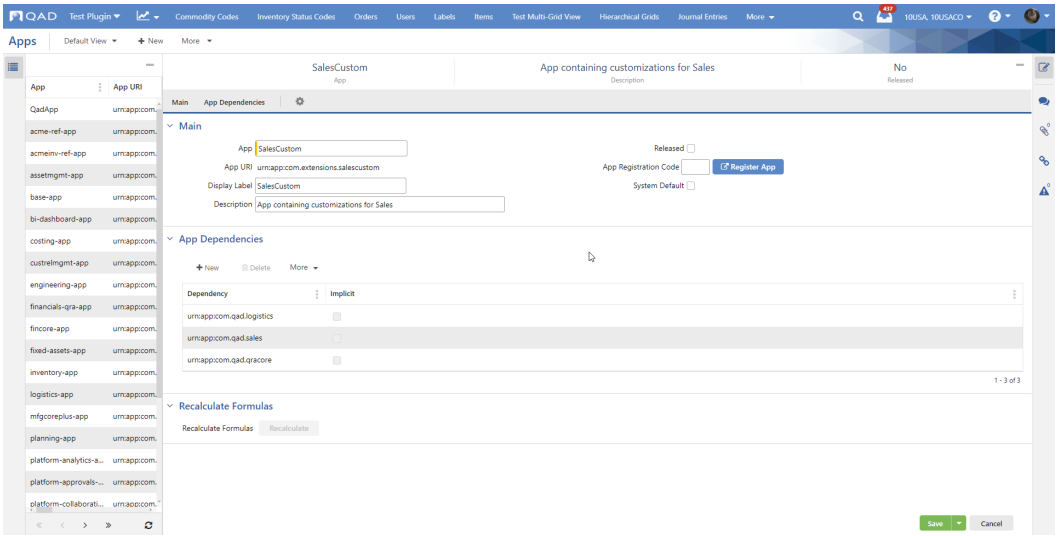
Steps

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.

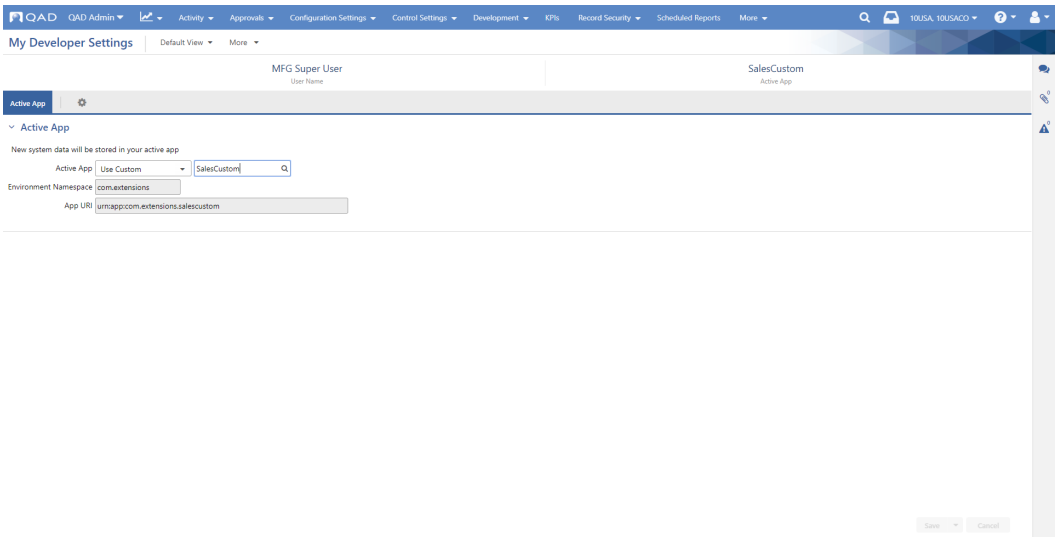
Our new app will have a dependency on the standard sales app, on the logistics app (because Sales Orders use definitions in logistics), and on qracore. The dependency on qracore will automatically be added when you save the app with the other dependencies in place.

Launch **Apps** from the menu and fill the necessary fields.



Step 2: Set the new app as default for yourself

Launch **My 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.extensions.salescustom

                    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:

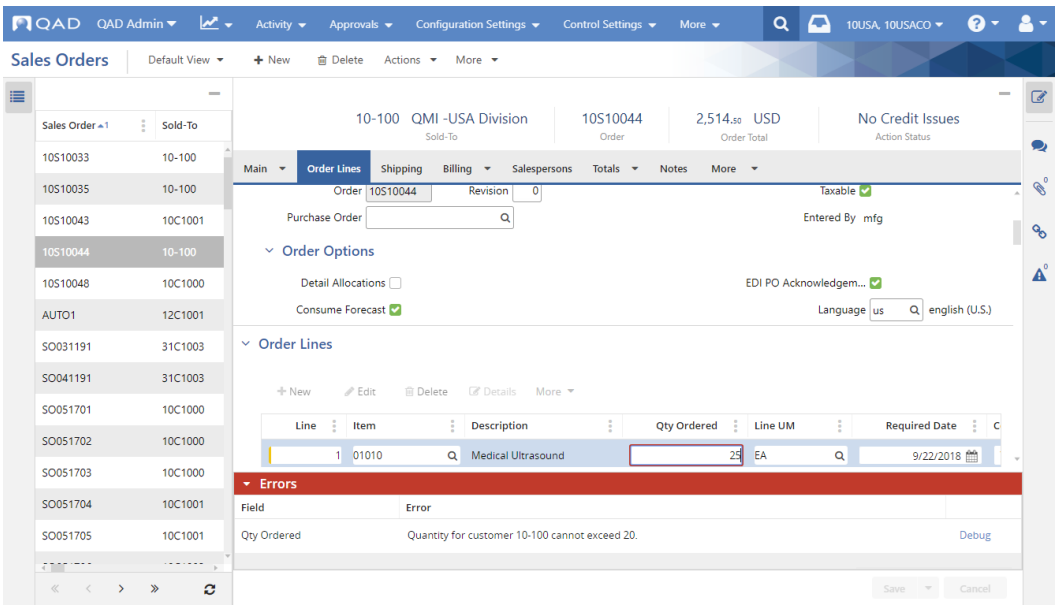
yab dev-<extractfolder>-code-register

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


QAD
QAD Admin
Activity
Approvals
Configuration Settings
Control Settings
More
10USA, 10USACO
Sales Orders
Default View
New
Delete
Actions
More
Sales Order
Sold-To
10S10033 10-100
10S10035 10-100
10S10043 10C1001
10S10044 10-100
10S10048 10C1000
AUTO1 12C1001
SO031191 31C1003
SO041191 31C1003
SO051701 10C1000
SO051702 10C1000
SO051703 10C1000
SO051704 10C1001
SO051705 10C1001
10-100 QMI -USA Division
Sold-To
10S10044
Order
2,514.50 USD
Order Total
No Credit Issues
Action Status
Main
Order Lines
Shipping
Billing
Salespersons
Totals
Notes
More
Order 10S10044
Revision 0
Taxable
Purchase Order
Entered By mfg
Order Options
Detail Allocations
EDI PO Acknowledgem...
Consume Forecast
Language us english (U.S.)
Order Lines
New
Edit
Delete
Details
More
Line
Item
Description
Qty Ordered
Line UM
Required Date
1 01010 Medical Ultrasound 25 EA 9/22/2018
Errors
Field
Error
Qty Ordered
Quantity for customer 10-100 cannot exceed 20.
Debug
Save
Cancel

Example 4: Extend Countries BC with an embedded BC (Capital, Population, Currency Code) and add OOABL validation for Currency Code

Table of Contents

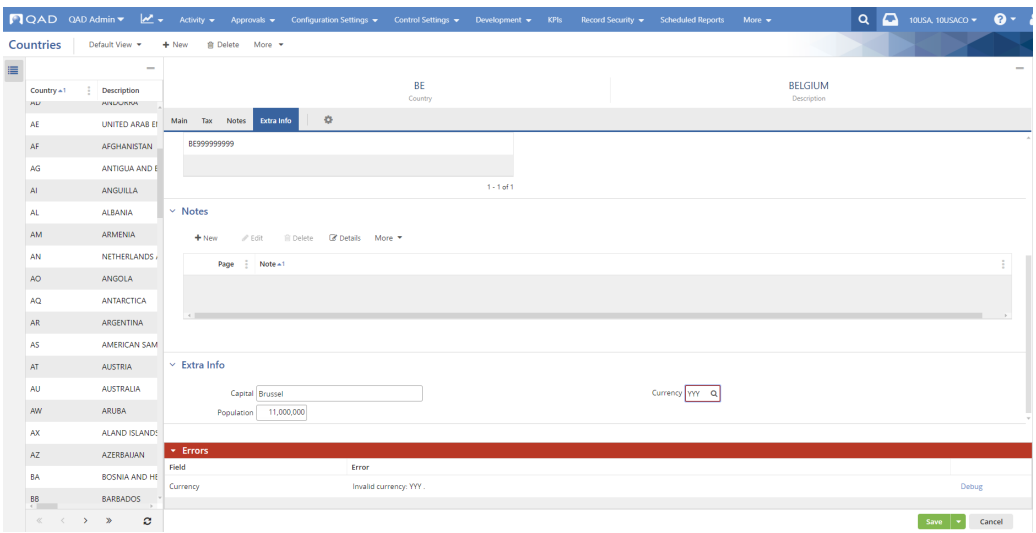
Overview

In this example, we will:

- add extra information to "Countries" (create an embedded BC which is aimed to extend ICountry)
- have a lookup on Currencies (lookup relationship)
- 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.

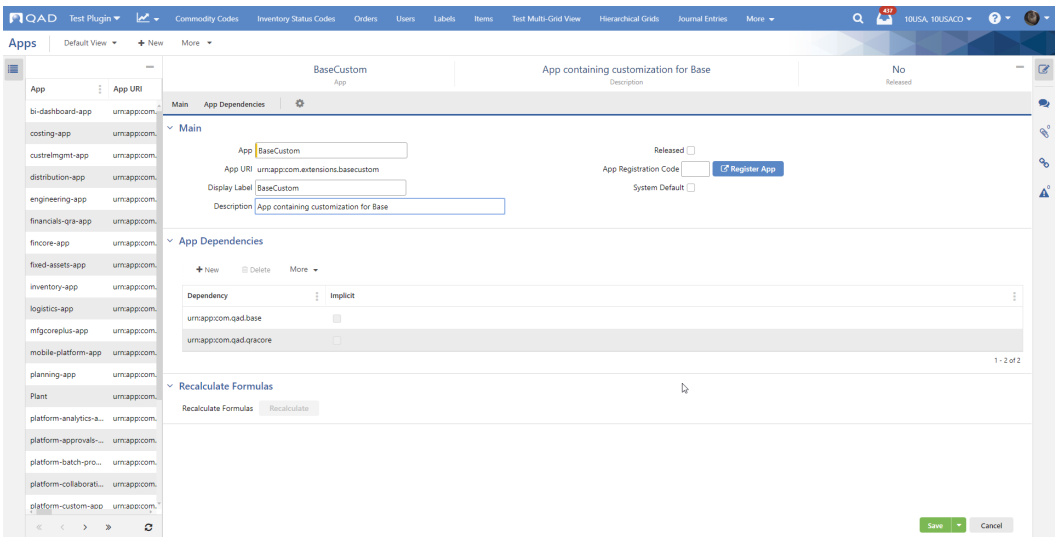
Steps

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.

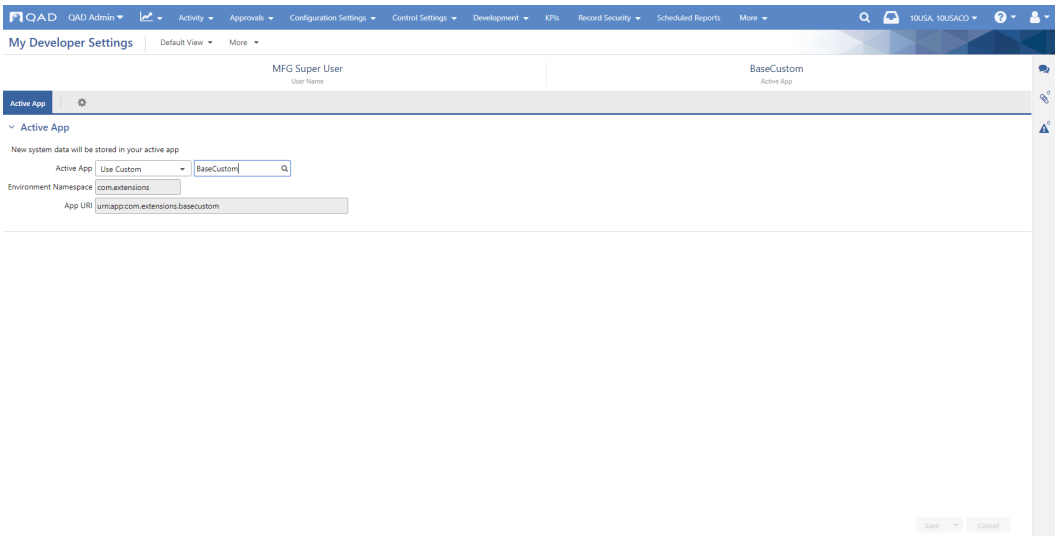
Our new app will have a dependency on the standard base app and on qracore. Note the dependency on qracore will automatically be added when you save the app with the other dependencies in place.

Launch **Apps** from the menu search and fill the necessary fields.



Step 2: Set the new app as default for yourself

Launch **My Developer Settings** from the menu search 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>

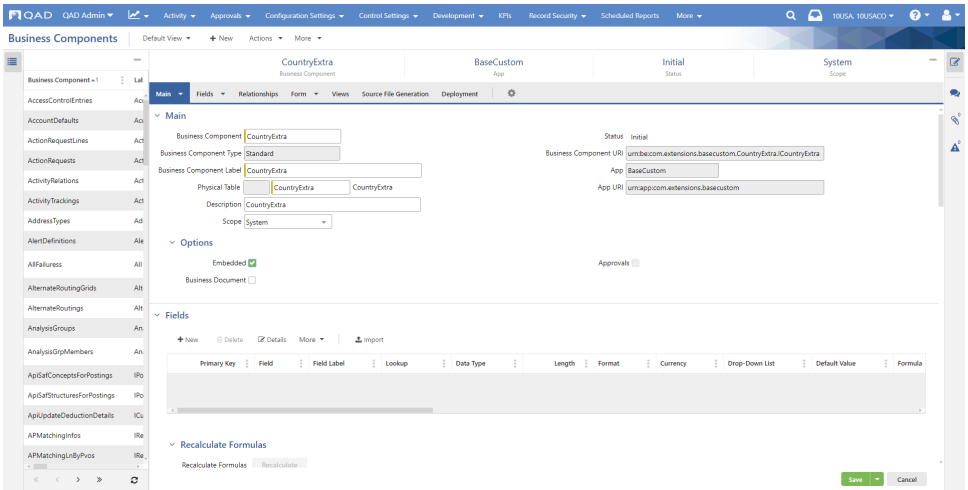
In our example, that is:

platform-extension.basecustom.dir=/dr01/qadapps/systest/basecustom

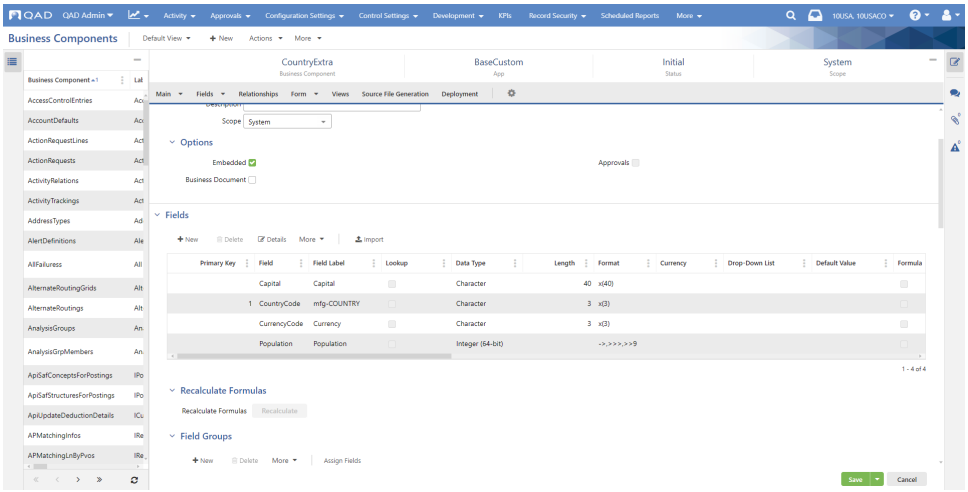
```
#------------------------------------------------------------------------
# configuration.properties
#
# This file can be used to overwrite any existing YAB configuration
# settings or to add/upgrade new packages to an environment.
#
# Example: Overwriting an existing property
# netui.telnet.user=odnetui
# 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 an embedded BC to extend ICountry

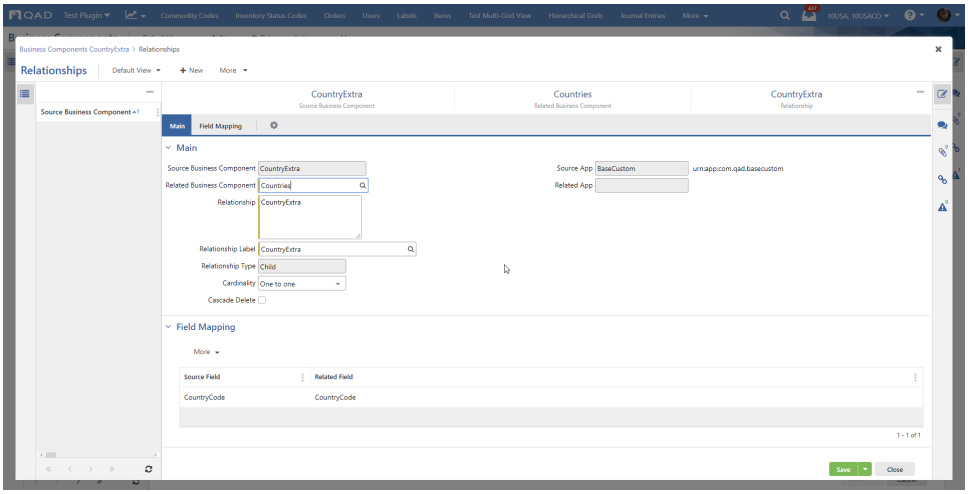
1. Open **Business Components** and create a new one.
2. **Main** panel: fill in the Main panel as follows (make sure the **Embedded** checkbox is selected).



3. **Fields** panel: fill in the Fields panel as follows. Note that we also need to add CountryCode, otherwise we won't be able to link this embedded BC to the main component.



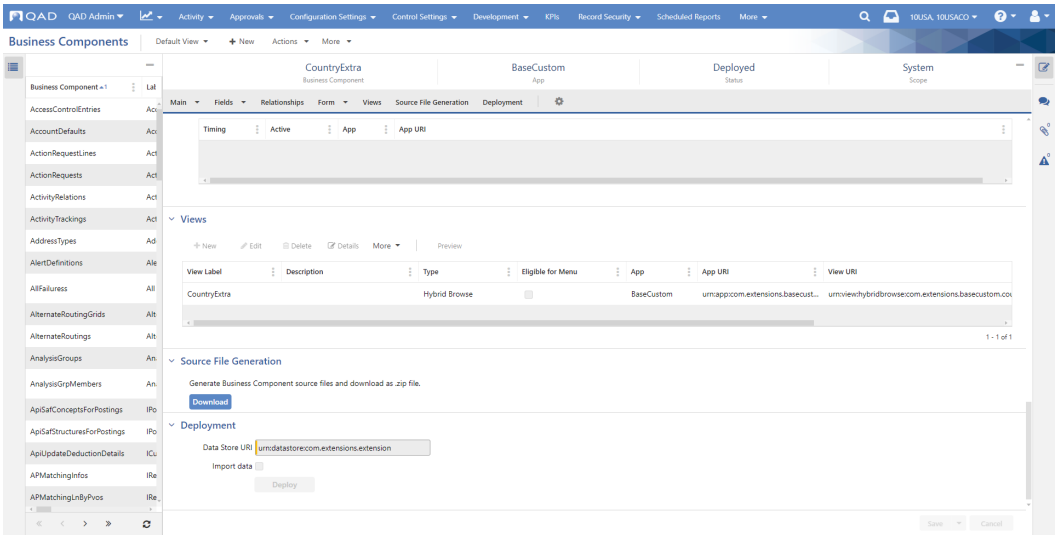
4. **Save**.
5. **Relationships** panel: now define the child-parent relationship between the embedded BC and Countries.



6. **Save** and **Close** the Relationships pop-up window, and then **Save** the business component.

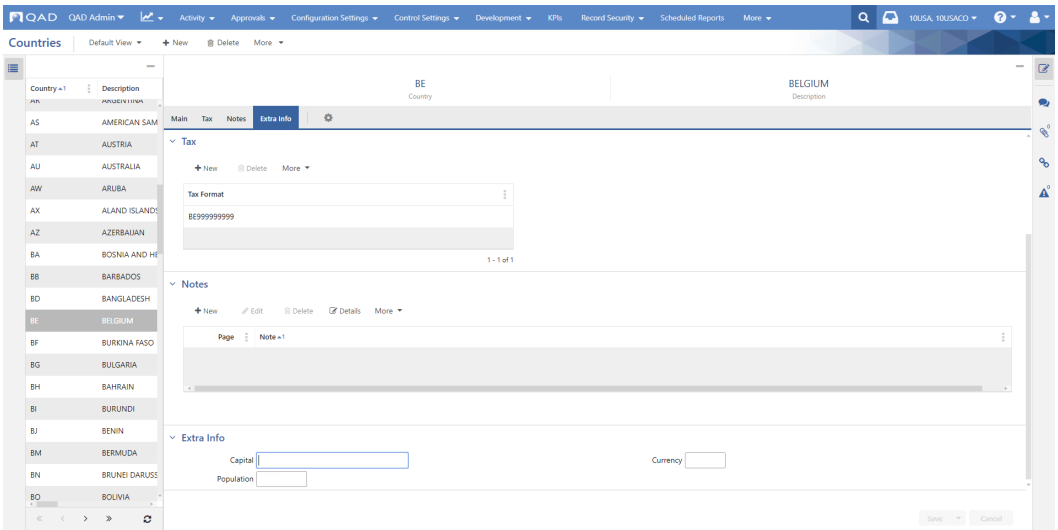
Step 6: Deploy the created embedded BC

Open the **Deployment** panel, select a data store, and then click **Deploy**.



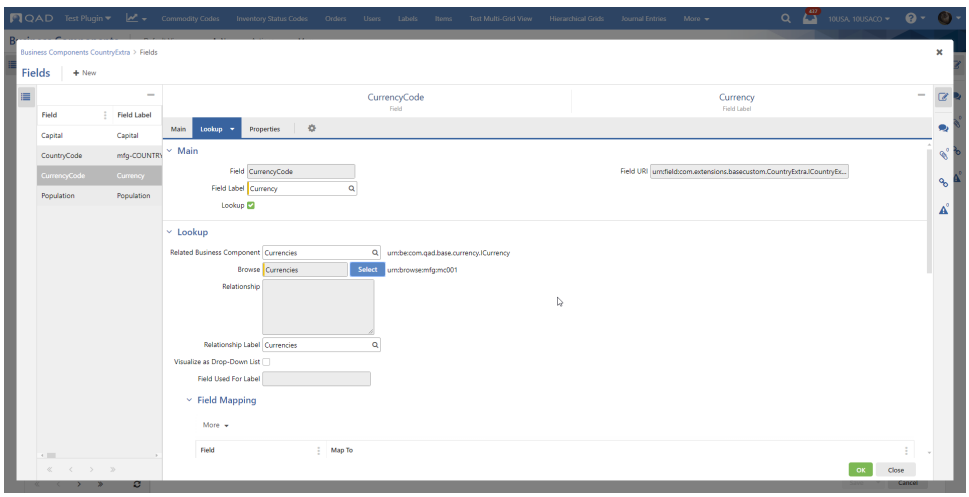
Step 7: Check it out

Run **Countries** from the menu search, scroll down and see the extra fields.



Step 8: Define the lookup for Currency

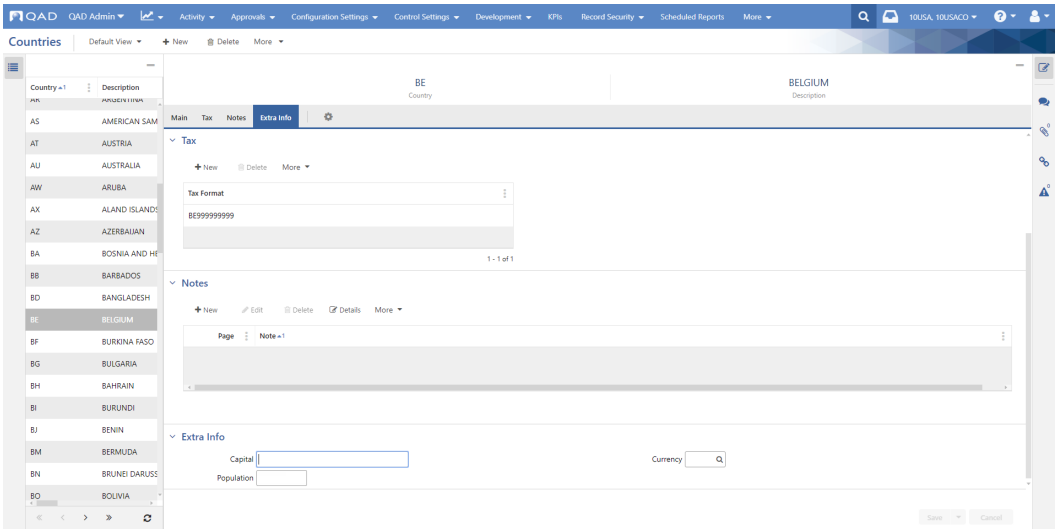
1. Run **Business Components** and open your embedded BC.
2. Open the **Fields** panel, select the CurrencyCode field, and then click **Details**.
3. Select the **Lookup** checkbox and define the lookup relationship to Currencies as follows.



4. Click **OK**, **Close**, and then **Save** again on the Business Components screen.

Step 9: Check it out again

Run **Countries** from the menu search, 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.

The following example is using a service to retrieve the currency code using a API. The statement used for this is: "BaseServices:GetCurrency():GetCurrencyByCode(input buf::CurrencyCode)". This means

- you need to know the **app** where the service is located (Base)
- there needs to be a corresponding 'get' method in <App>Services to **instantiate** the correct class (GetCurrency(), but there are cases where this is not defined)
- there needs to be a method that returns the **dataobject** and the field(s) you are interested in (GetCurrencyByCode but there are cases where this is not defined)

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:

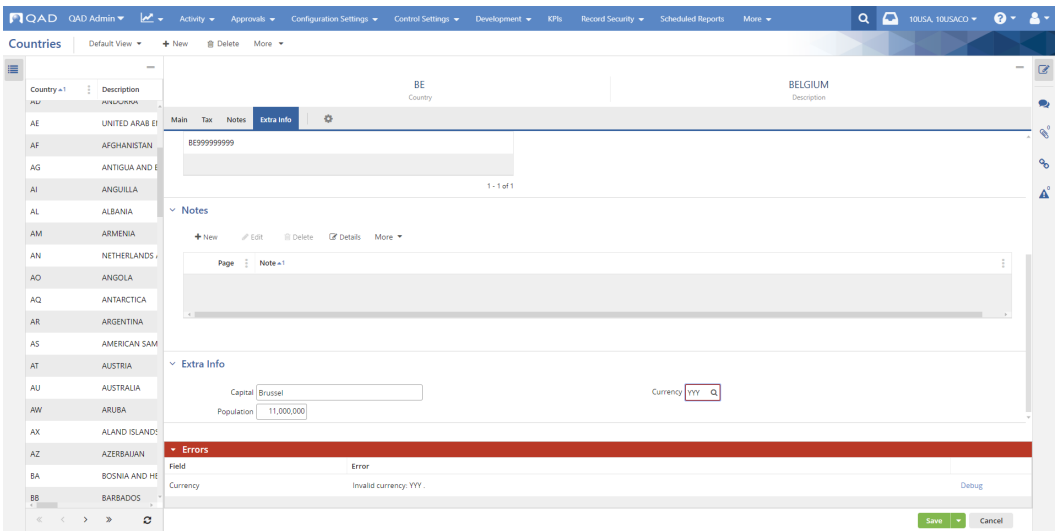
yab dev-<extractfolder>-code-register

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



App Security

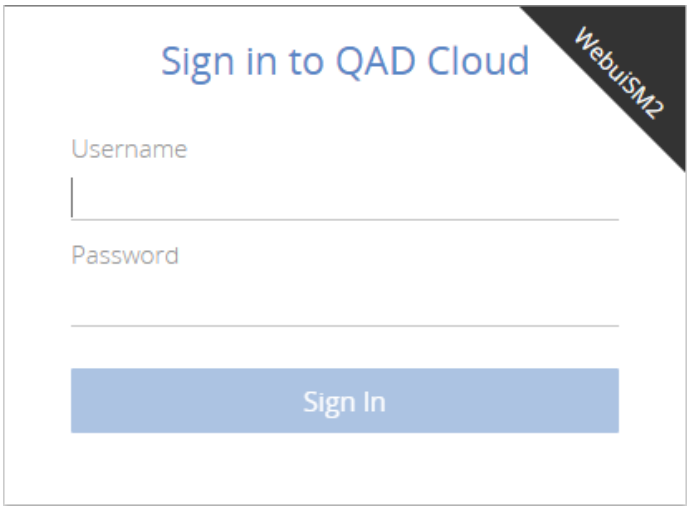
This section covers a brief overview of QAD Enterprise Platform security features as it relates to App development.

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.

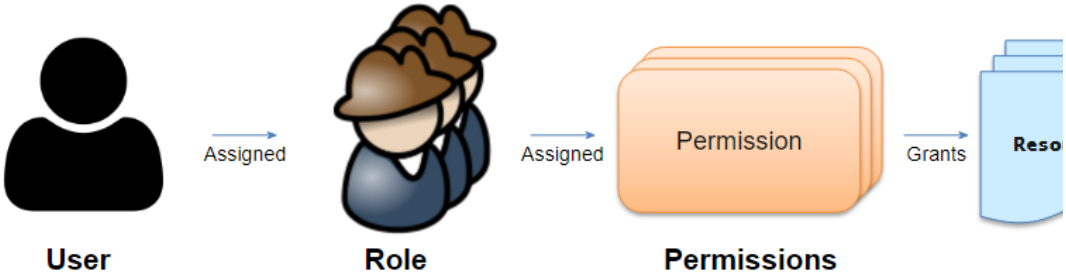
Session Management

QAD Enterprise Platform provides a session management service. This service should be transparent to App developers.

Role Based Access Control

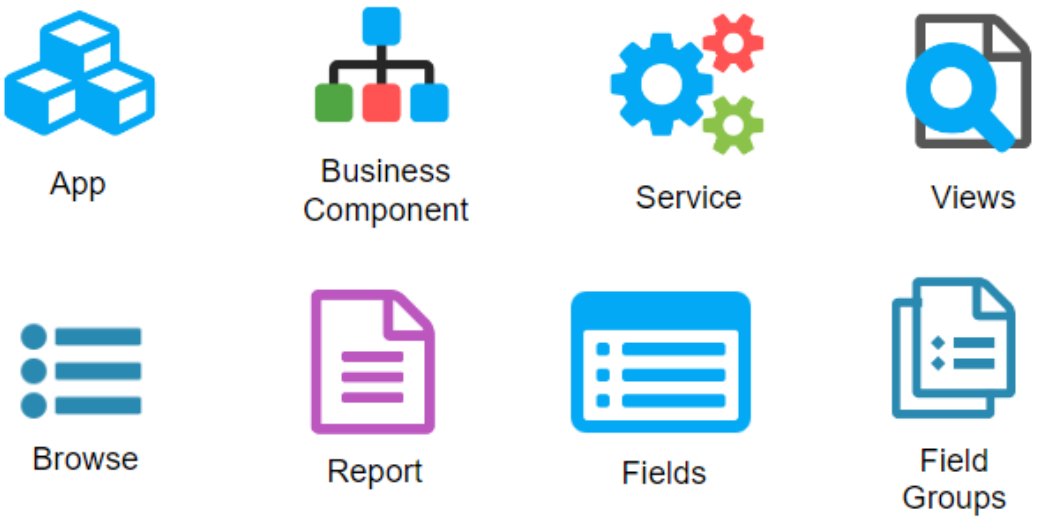
QAD Enterprise Platform supports a Role based access control model. In it Roles are defined at the system level and users can be assigned multiple Roles.

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

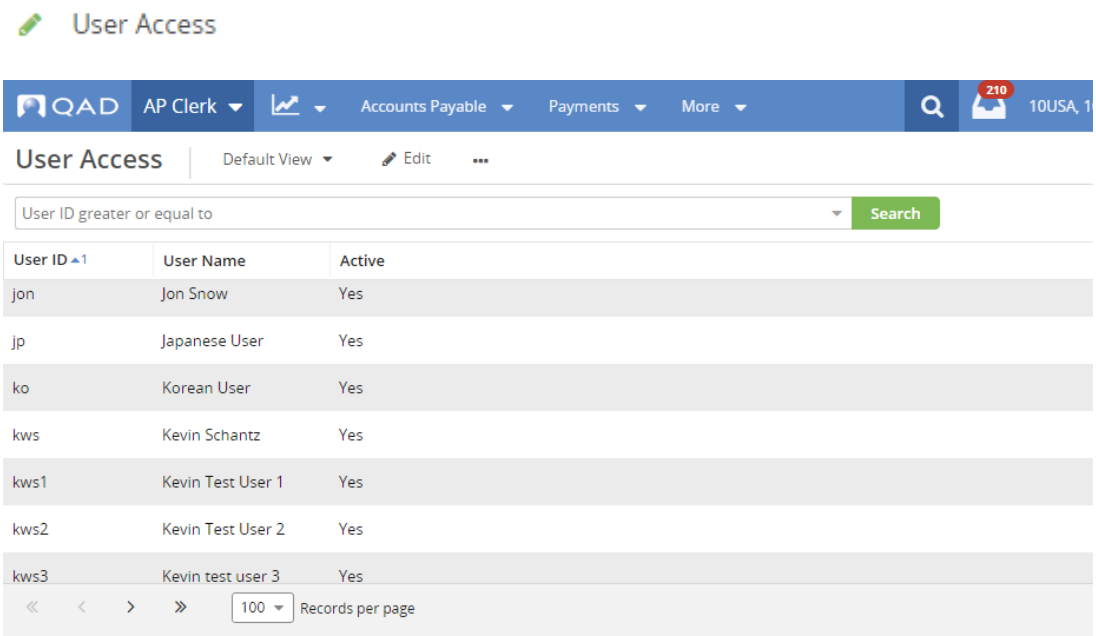
Users can be managed by a system administrator through the .NetUI application or externally defined in a directory service such as LDAP or Active Directory.

- Users defined by administrator
- Must have webui_user role for CI access
- Must be assigned roles for their specific job function

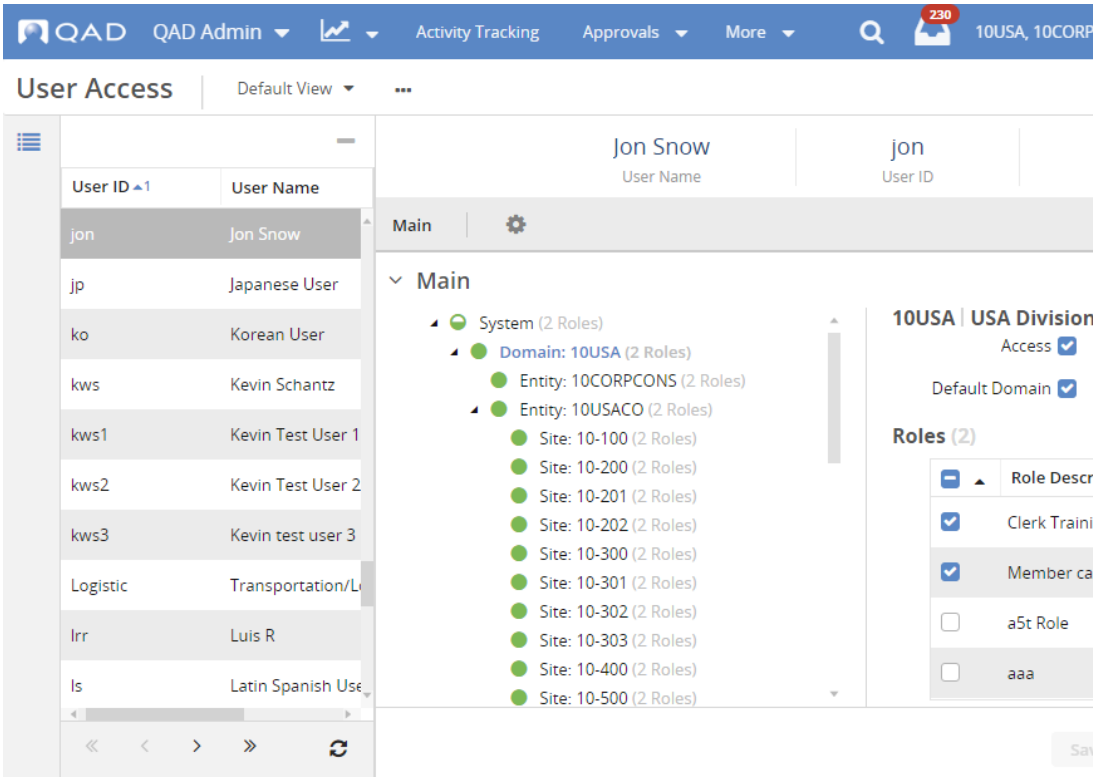
User Access

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.

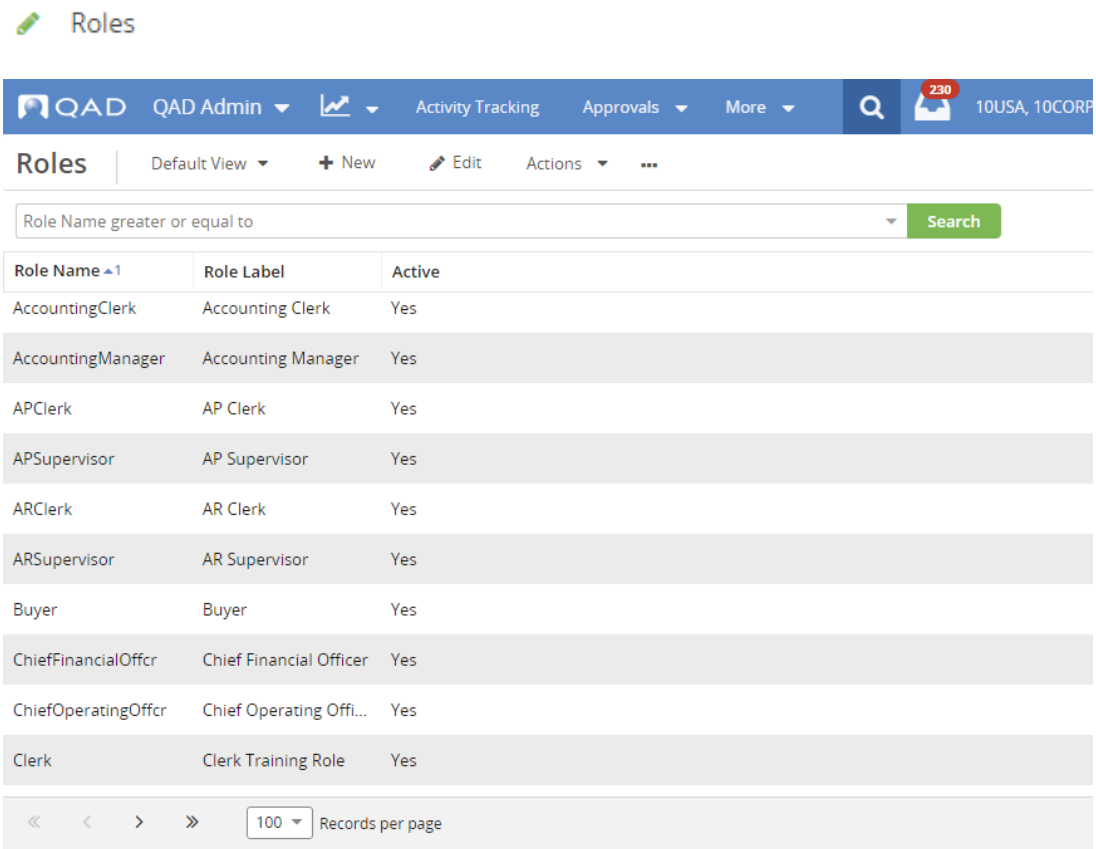

QAD
QAD Admin
Activity Tracking
Approvals
More
230
10USA, 10CORP
User Access
Default View
User ID
User Name
jon
Jon Snow
jp
Japanese User
ko
Korean User
kws
Kevin Schantz
kws1
Kevin Test User 1
kws2
Kevin Test User 2
kws3
Kevin test user 3
Logistic
Transportation/L
lrr
Luis R
ls
Latin Spanish Use
Jon 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.

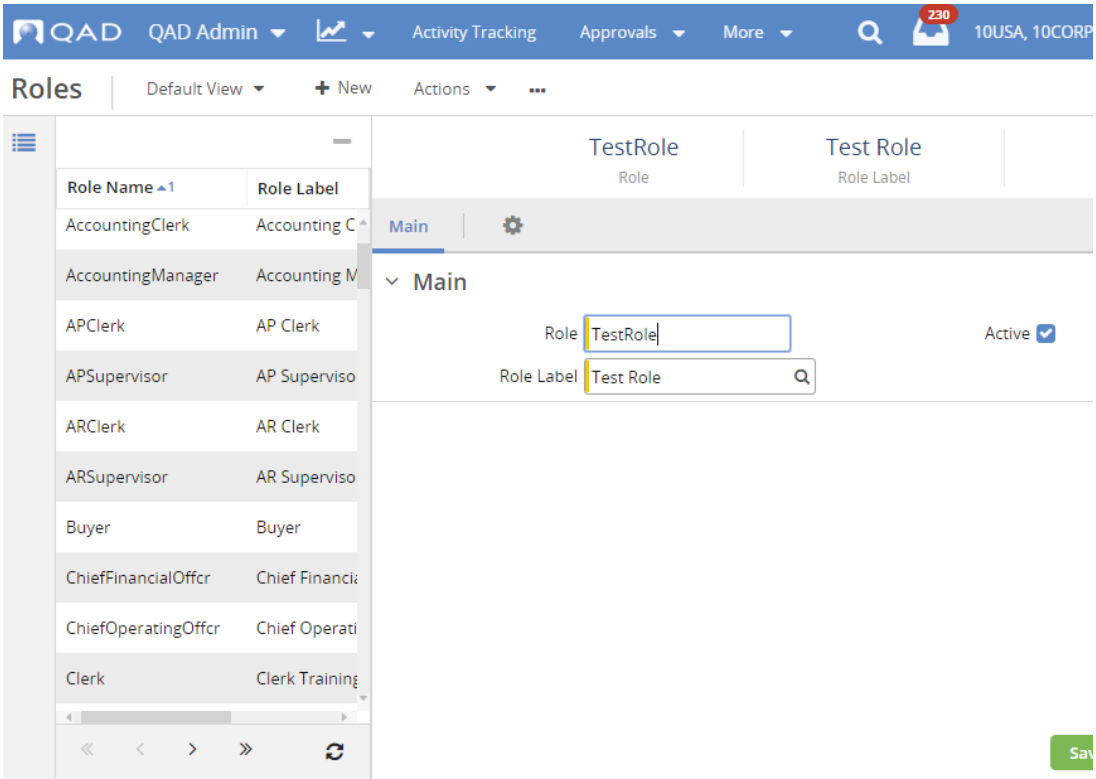
- Roles are defined at the system level
- Roles & role menus delivered by QAD are samples only
- QAD reserves the right to change sample role menus or permissions
- Customers should create their own roles
- Customers may copy existing role menus

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.

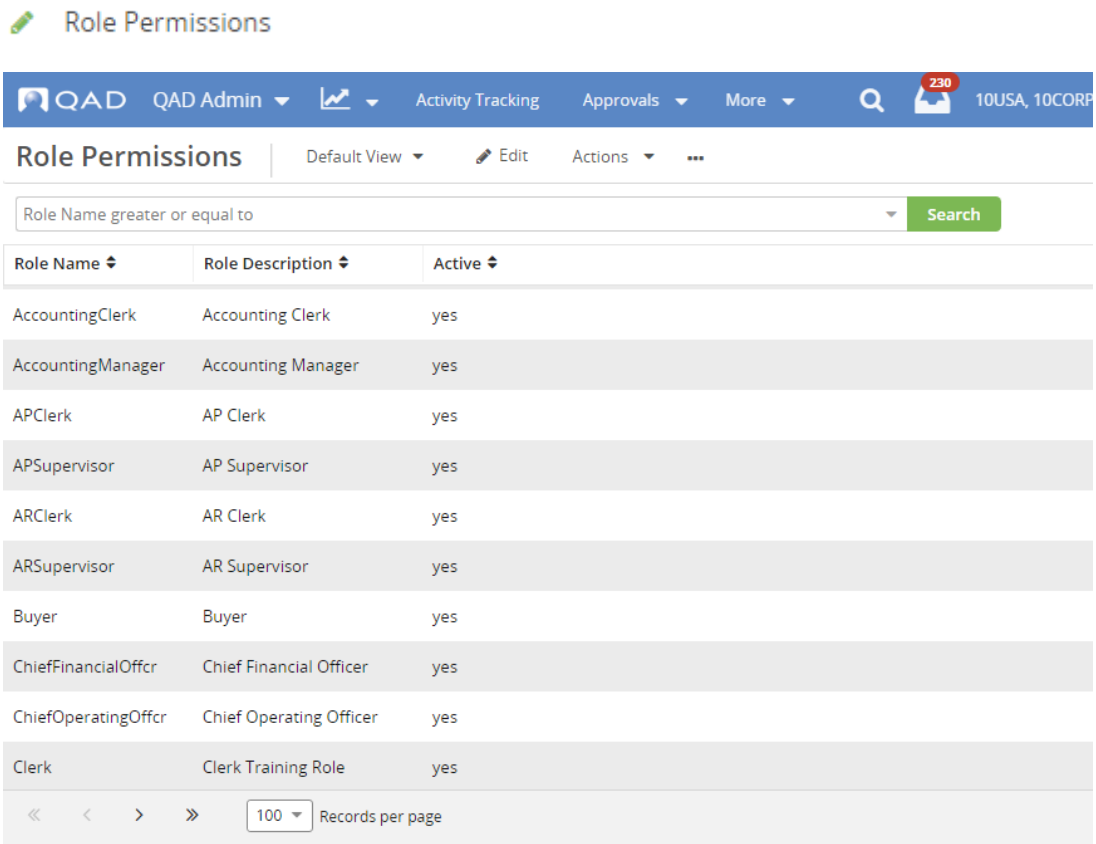

QAD
QAD Admin
Activity Tracking
Approvals
More
230
10USA, 10CORP
Roles
Default View
New
Actions
Role Name
Role Label
AccountingClerk
Accounting C
AccountingManager
Accounting M
APClerk
AP Clerk
APSupervisor
AP Superviso
ARClerk
AR Clerk
ARSupervisor
AR Superviso
Buyer
Buyer
ChiefFinancialOffcr
Chief 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.

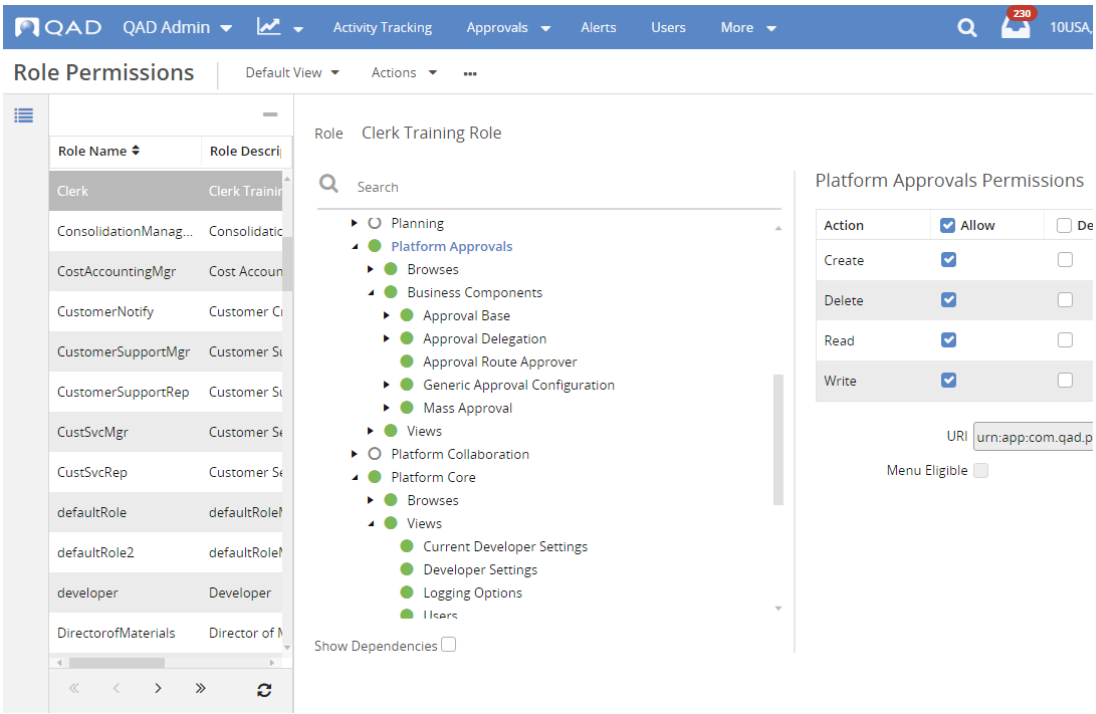
Resource Type	Permissions
Apps	Approve, Create, Delete, Read,
Business Components	Approve, Create, Delete, Read,
Browses	Read
Dashboard & KPIs	Read, Write, Delete
Reports	Read
Services	Create, Delete, Read, Write
Fields	Read, Write
Field Groups	Read, Write
Views	Read

Roles have permissions assigned through Role Permissions screen.

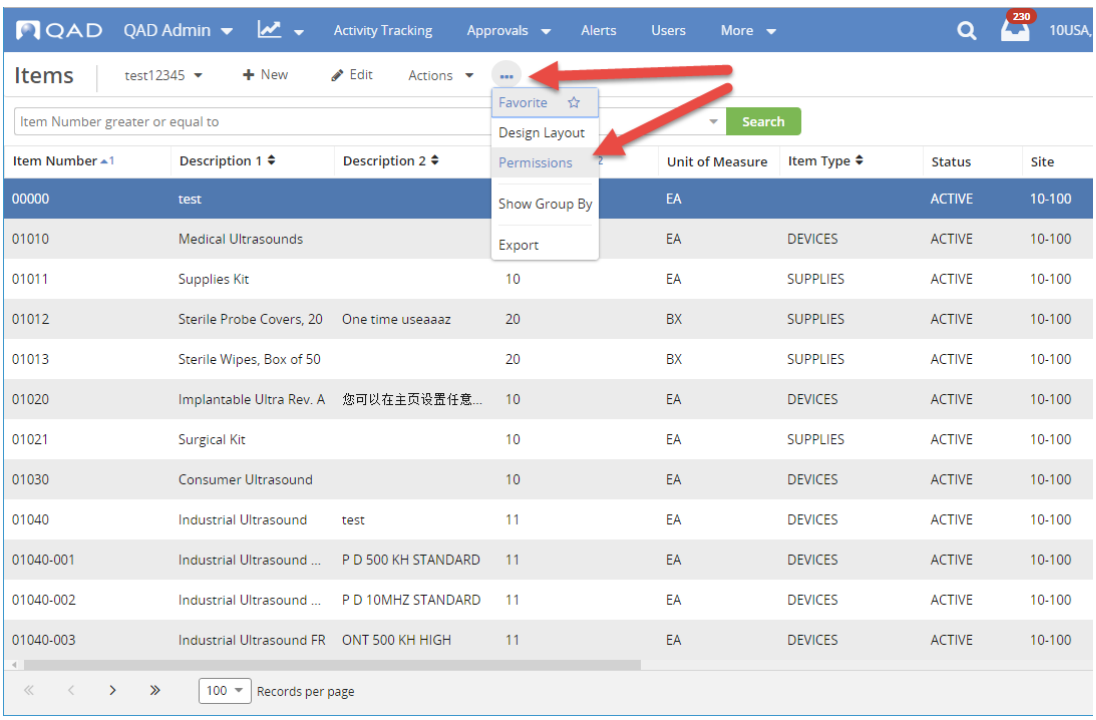


To assign/change permissions

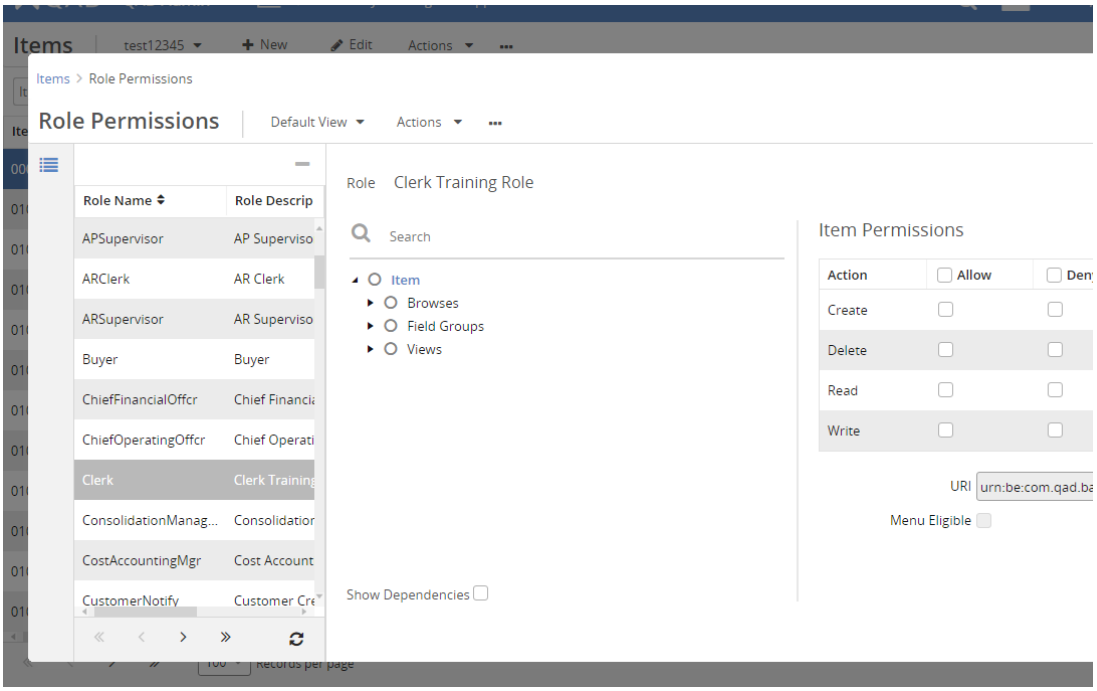
- Select Role (double click)
- Expand tree view
- Set desired permission
- Save changes



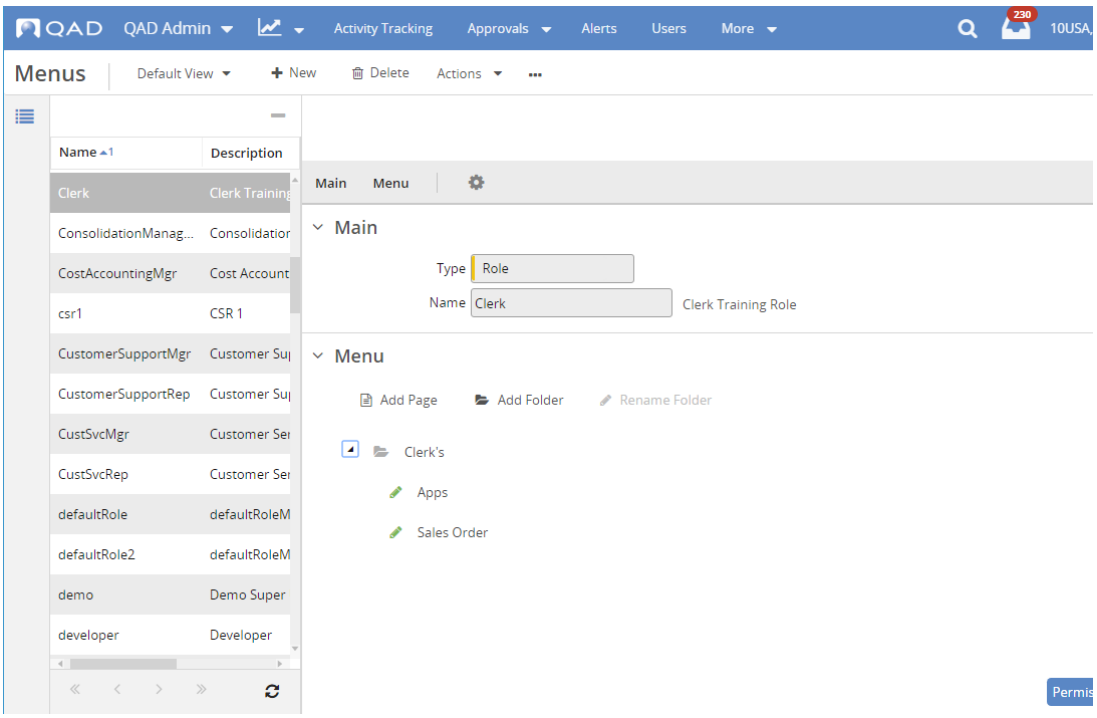
Role Permissions should be granted by a system administrator. There are two other was to launch Role Permissions depending on the context, from the drop down menu:



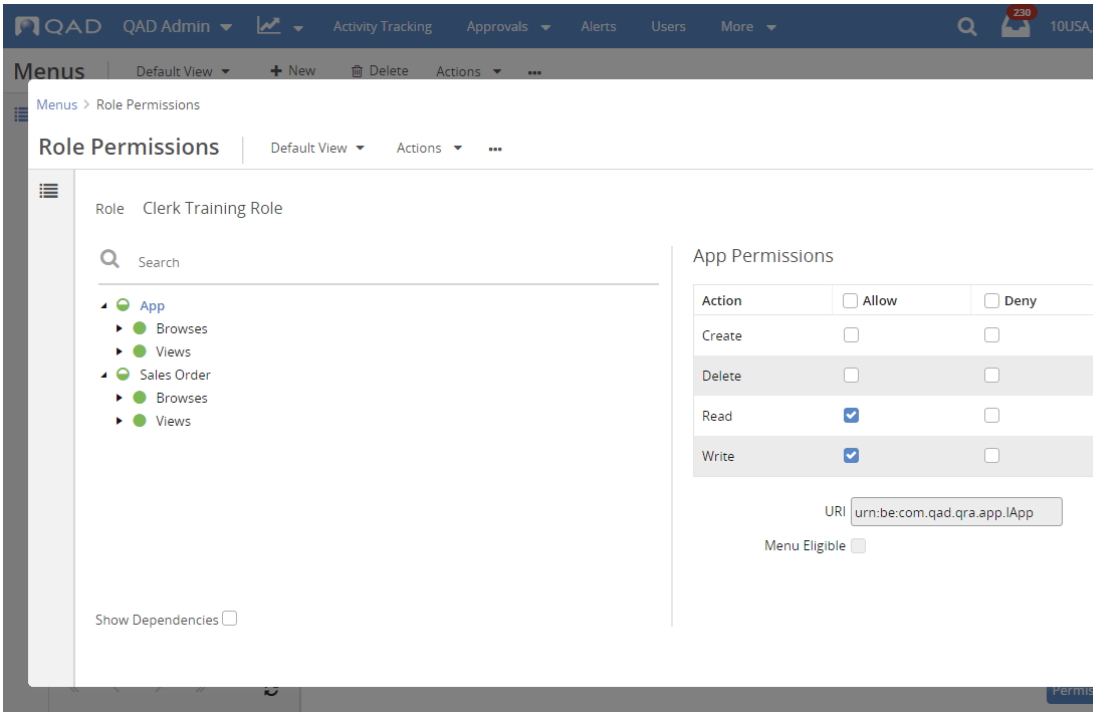
Only the resources in that context will be available to assign permissions



Finally through defining a Role Menu in the "Menus" screen. Click on Permissions

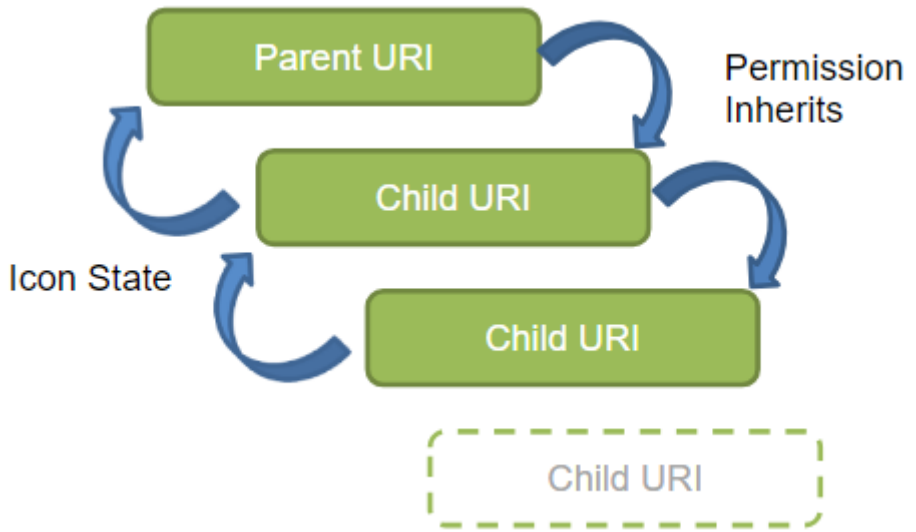


Permissions are only for the items related to the menu

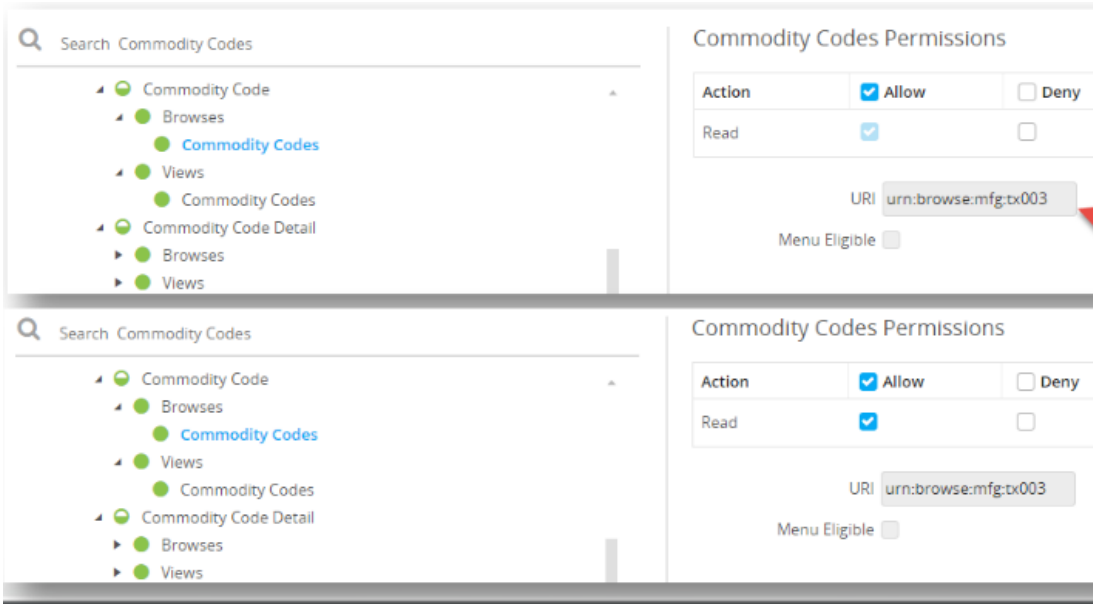


Effective and Explicit Permissions

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

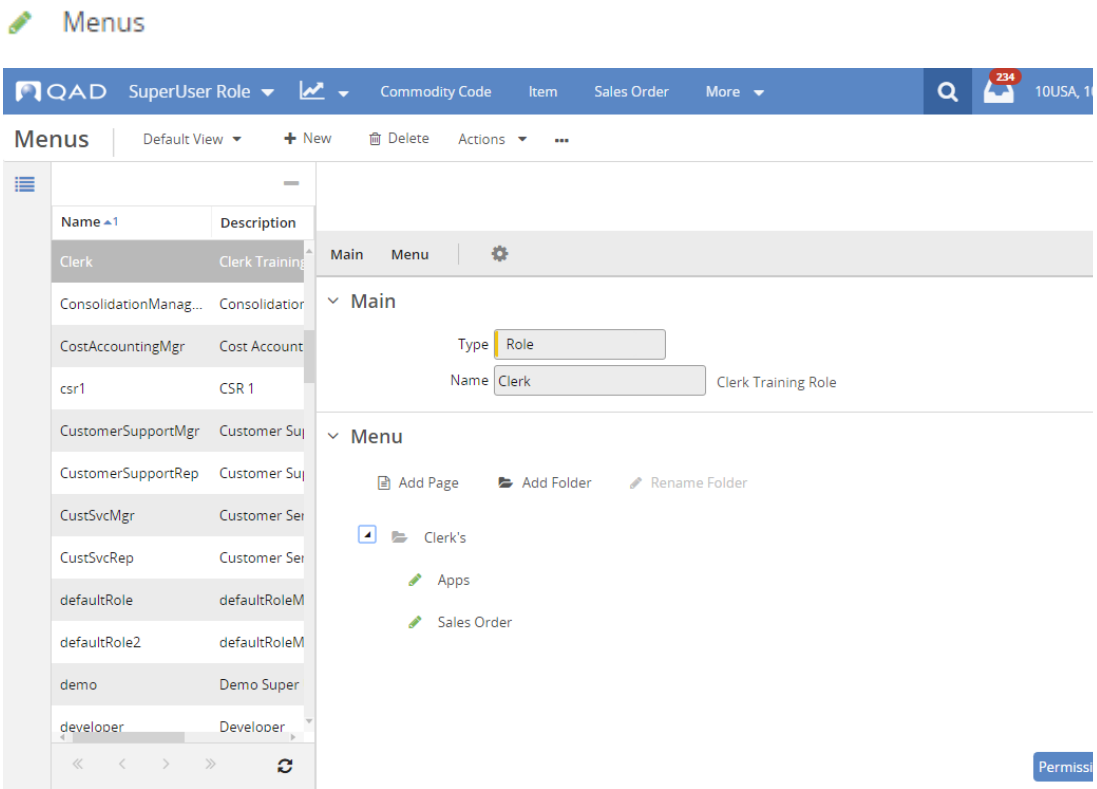
Role Menus

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

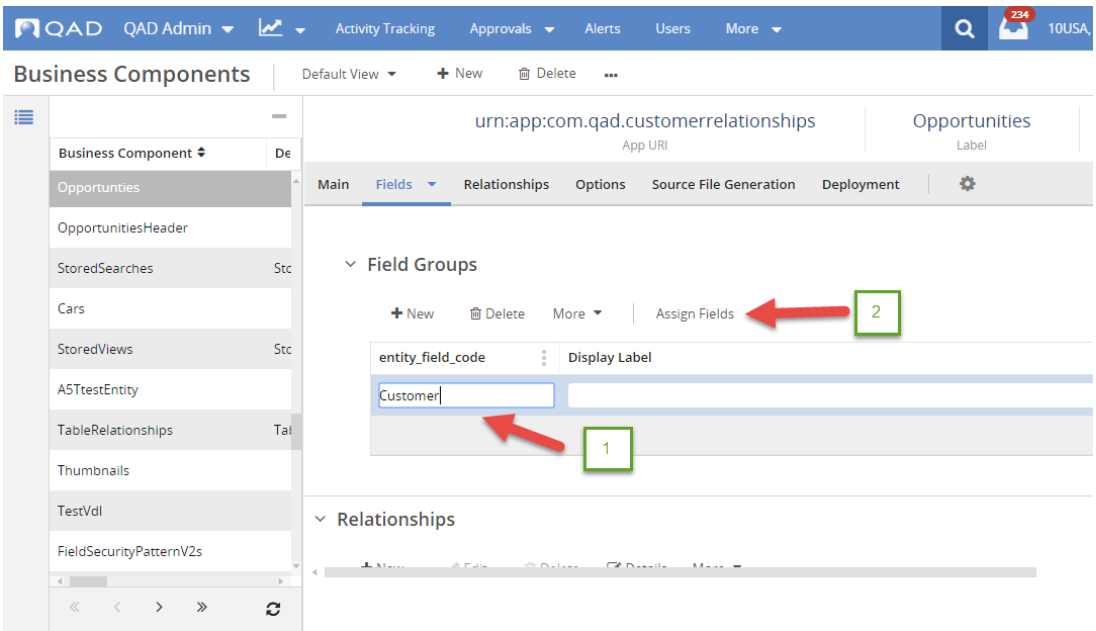
QAD Enterprise Platform Applications makes it possible to control permissions at the field level. Fields are hidden to an end users that don't have permissions to read it. Field security is enforced at the UI and API level. Calling APIs may return null values, developers are expected to handle it and avoid saving "null" values for users that don't have read or write privileges.

Field Groups

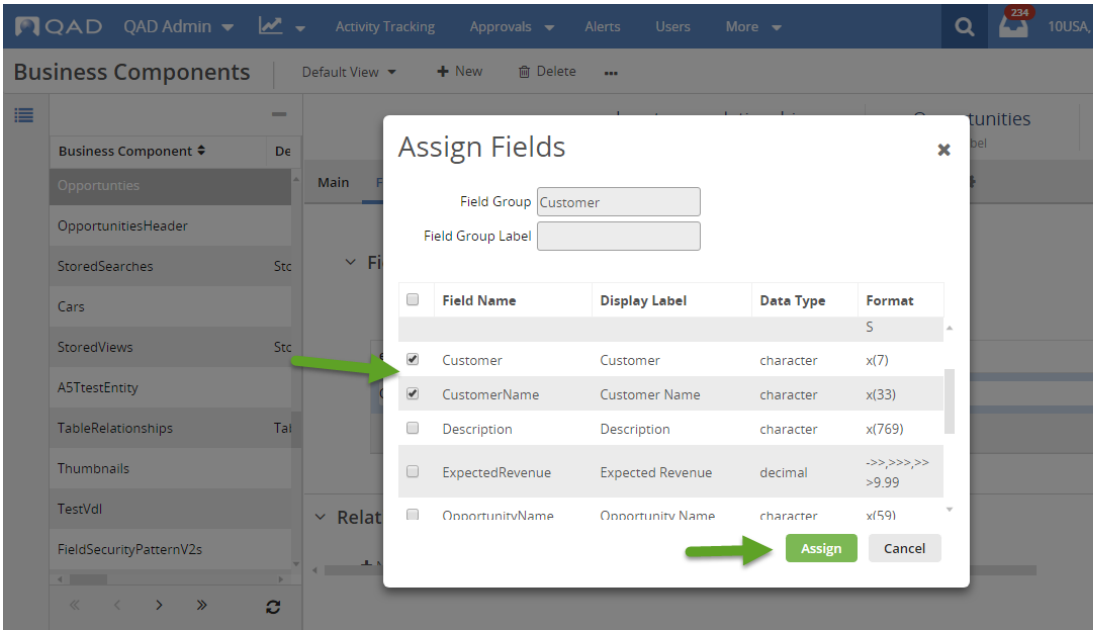
Field Group definitions are not required but are important for system administrators to quickly identify and set permissions to a group of fields. It is important to define "Field Groups" as part of the Business Component definition.

- 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.

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 using security settings

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.

Collaboration Administration

Collaboration administration includes managing activity tracking and alerts.

Activity Tracking

Introduction

On the Activity 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 on the Activity 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).

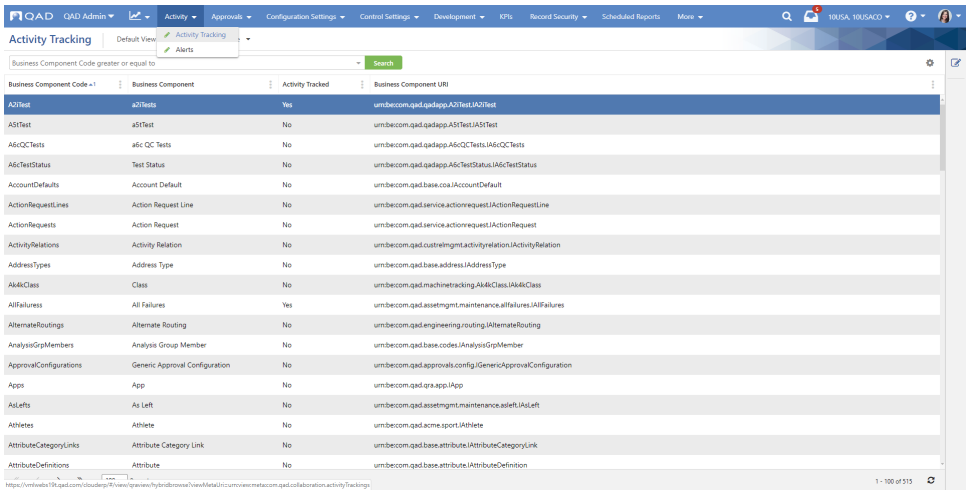
When a business component has activity tracking enabled for the first time, a system administrator must run a YAB command that will compile and load the triggers into the database. The YAB command is as follows:

```
yab activity-feed-update
```

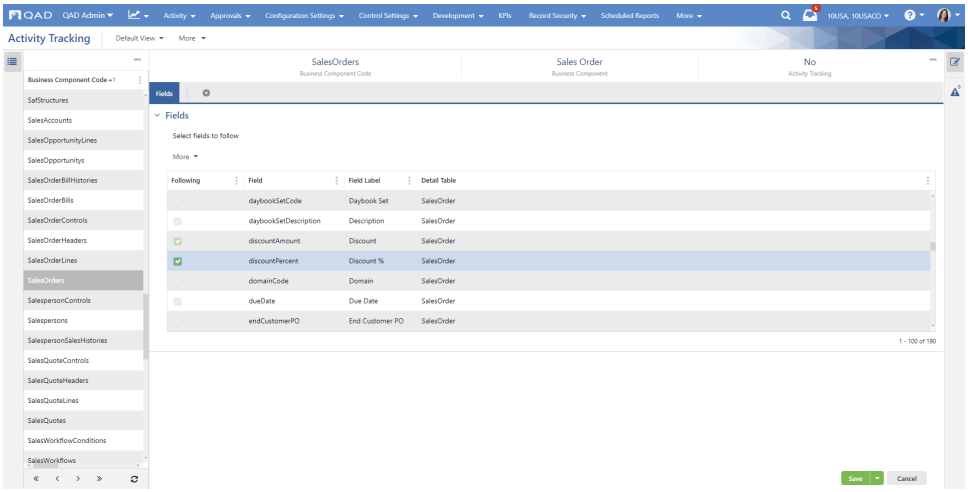
Setting up Activity Tracking for the Business Component fields

To set up Activity Tracking for the Business Component fields:

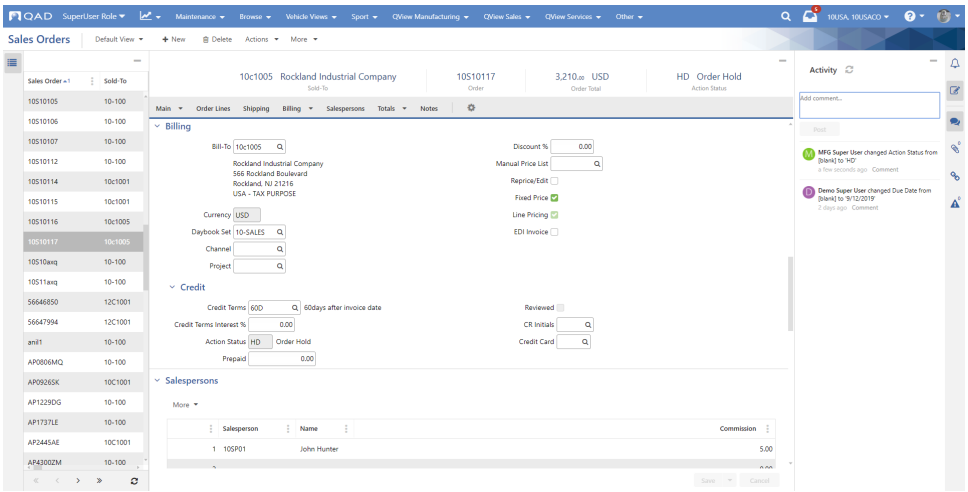
1. Click **Activity** on the menu bar, select **Activity Tracking**, and then double-click the Business Component.



2. In the **Following** column, you can choose which fields of the specific Business Component you want to track, including the extended fields, and then 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.



3. To check how it works, find the Business Component in the menu search, change the data in the tracked fields, and then click **Save**.
4. Open the **Activity** panel on the right side of the page by clicking the **Activity** icon, and then click the **Refresh** icon. You will see the changes you made on the **Activity** panel.

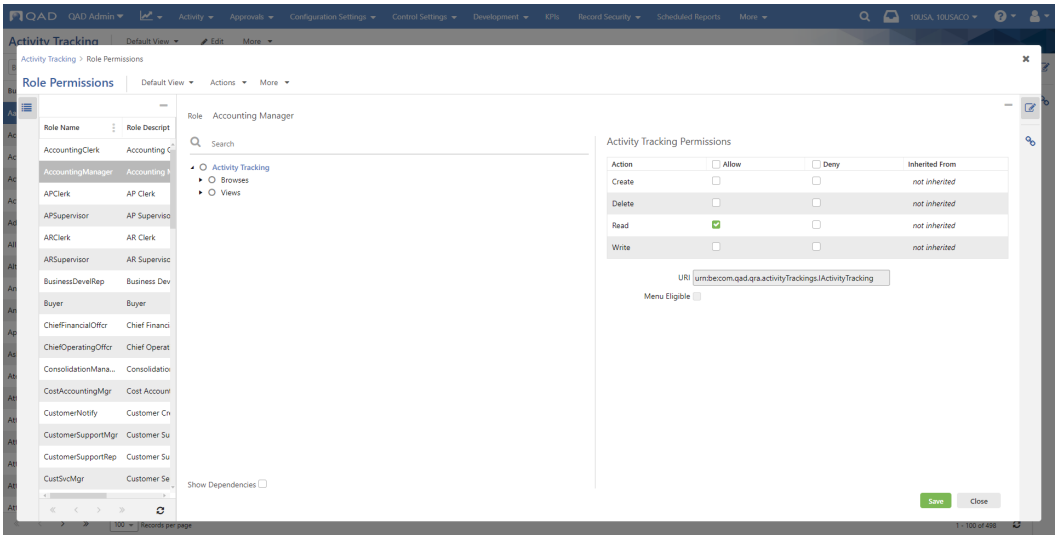


Setting up Permissions for Activity Tracking

Use Role Permissions to view settings for all roles in your system and to fine-tune permissions for roles you have created. The grid lists the actions for the resource that can be set to Allow or Deny, and if the permissions were inherited. If neither the Allow nor Deny checkbox is selected, the role inherits permissions from its parent resource.

To set up role permissions for Activity Tracking of the Business Component fields:

1. On the **Activity Tracking** page, 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**.


Activity Tracking > Role Permissions
Role Permissions
Default View
Actions
More
Role Name
Role Descript
AccountingClerk
AccountingManager
APClerk
AP Clerk
APSupervisor
ARClerk
AR Clerk
ARSupervisor
BusinessDevelRep
Buyer
ChiefFinancialOffcr
ChiefOperatingOffcr
CostAccountingMgr
CustomerNotify
CustomerSupportMgr
CustomerSupportRep
CustSvcMgr
Role Accounting Manager
Search
Activity Tracking
Browses
Views
Show Dependencies
Activity Tracking Permissions
Action
Allow
Deny
Inherited From
Create
Delete
Read
Write
not inherited
URI
urn:becom.qad.qra.activityTrackings.IActivityTracking
Menu Eligible
Save
Close

Alerts

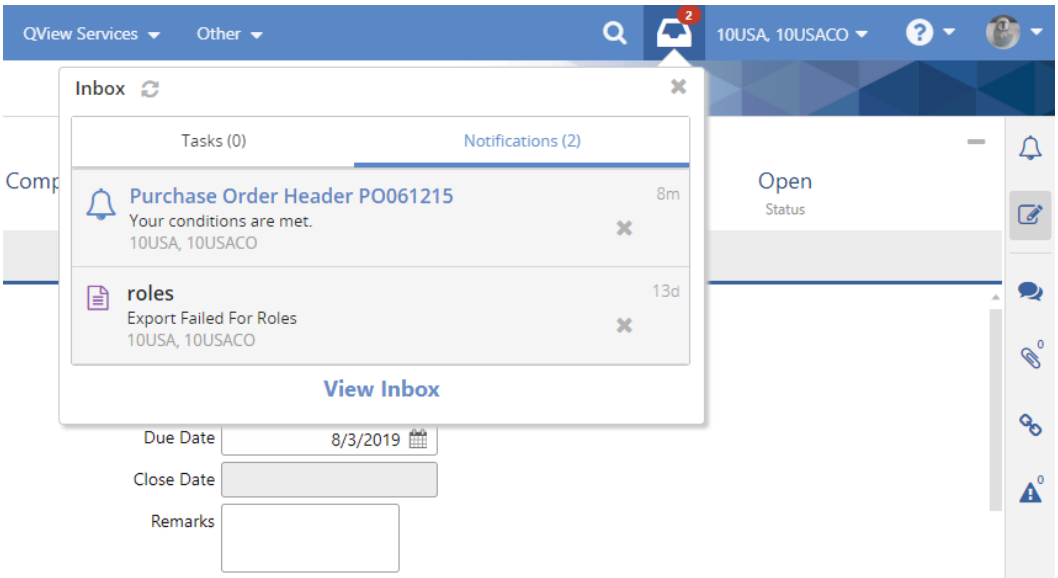
Alerts provide the capability to notify users when certain conditions are met. For example, you can create an alert so that if a purchase order's revision number changes, a notification is automatically sent to a user's Inbox or email address. Alerts can be created by users and by administrators, who 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.

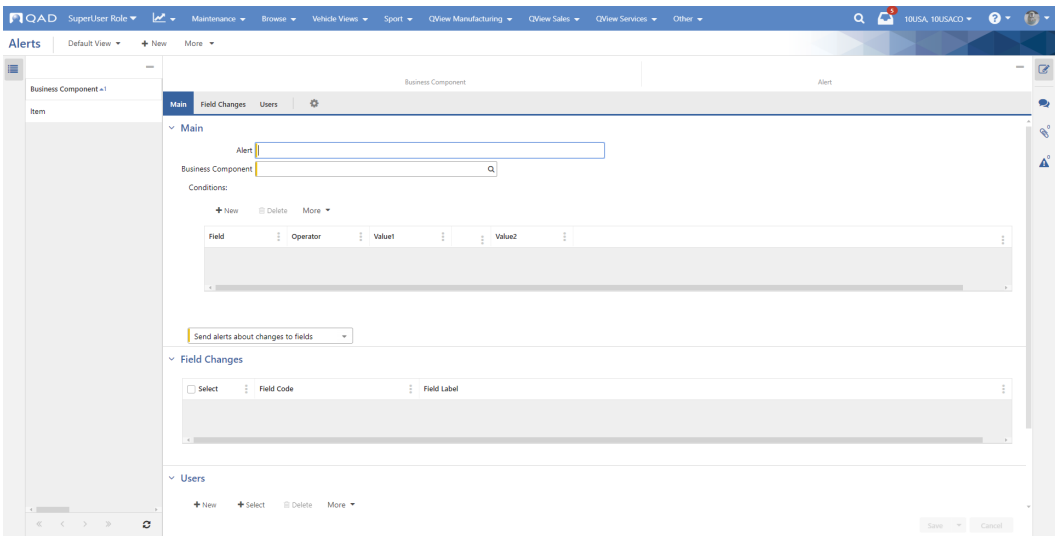
The Alerts pop-up window for users is identical to the Alerts view for administrators, except the Alerts view includes additional settings to select users for alerts. In the Alerts pop-up windows, users can see both the alerts that they have defined and the alerts defined for them by an administrator. Note that alerts defined by an administrator can only be changed by an administrator.

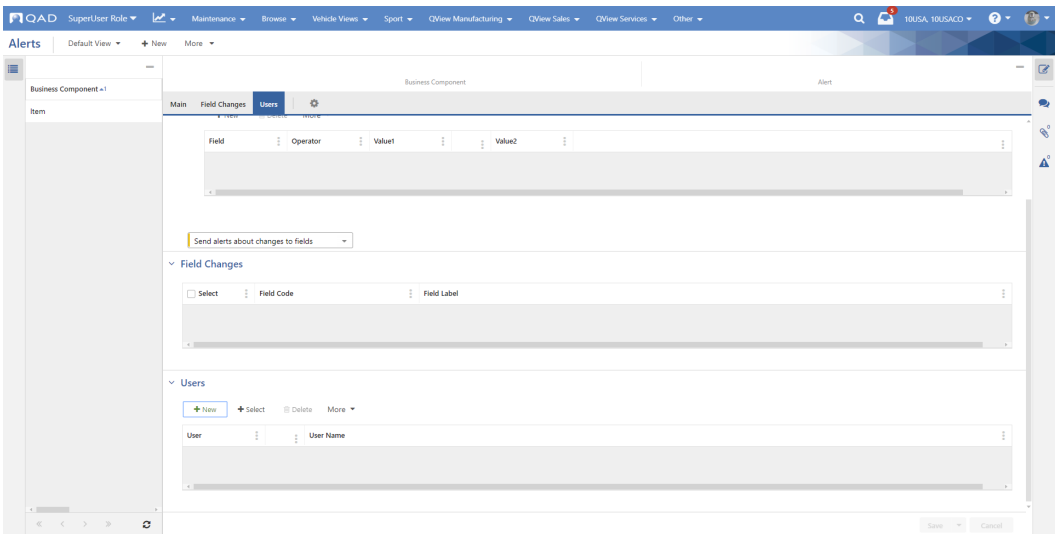


Alerts View

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 brief description for the alert.

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, including the extended fields.
- **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, including the extended fields.
- **Value2**: For operators that require two values, select from available values or click the toggle icon to select available fields, including the extended 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. Extended fields are also available in the list. 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.

You can add a literal user ID or a non-literal user, coming from a business component field. For the steps on how to add users to alerts, see Creating Alerts - Step by Step.

- Literal user ID. This way, you can send alerts to the user added from the Users lookup.
- Non-literal user. This way, you can send alerts to the user defined in a business component field, such as the Requester or Entered by field. For example, when a requisition becomes Out-of-Tolerance, the requester needs to take action on the requisition document. In this case, you can notify only the specific requester or Entered by user that is specified on the requisition document that the requisition has become Out-of-Tolerance.

Click **New** to select active users who will receive this alert.

User

Select a user or a field from the lookup.

Toggle Icon

Click the toggle icon to switch between a literal user ID (with the Users lookup) and a non-literal user (with the Fields lookup).

User Name

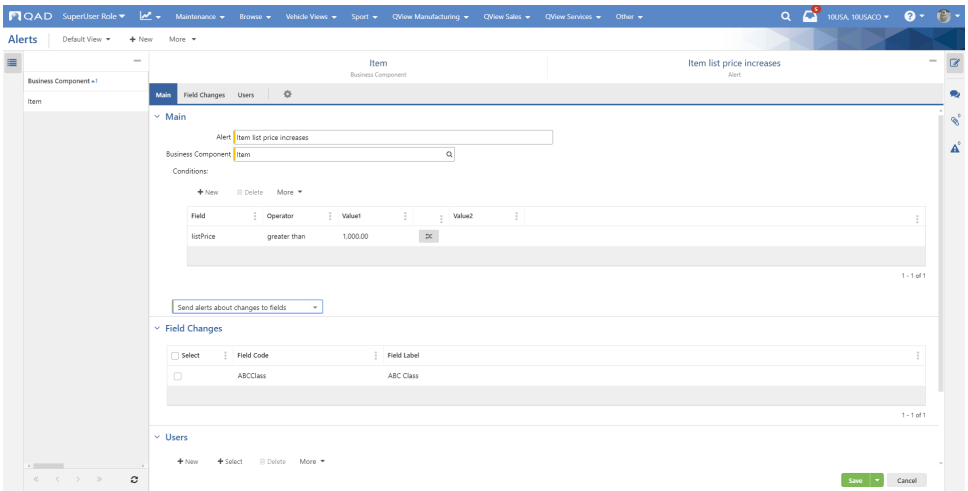
Displays the name of the user who will receive alerts. When a non-literal user is selected, the **User Name** column is blank.

Creating Alerts - Step by Step

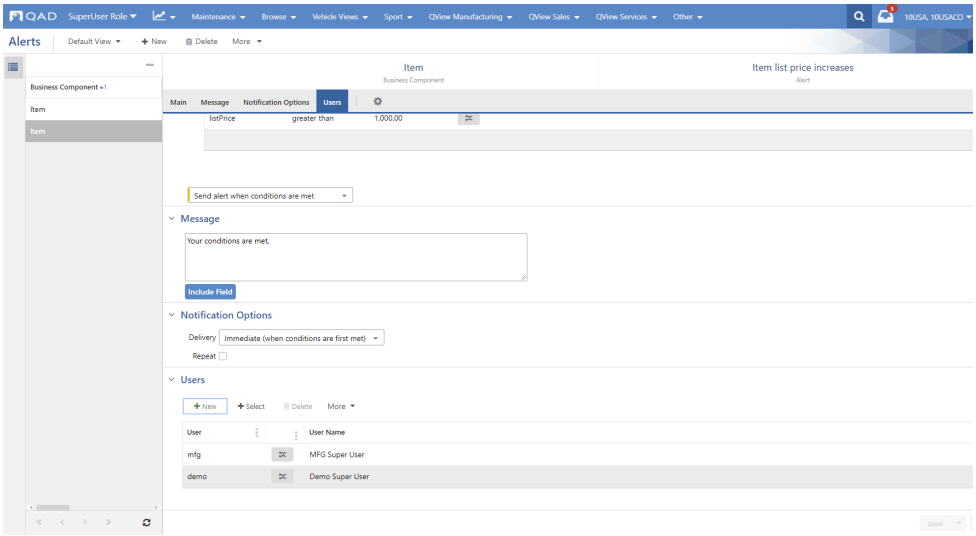
Before you can set alerts about field changes on a hybrid view form, you must enable activity tracking for those fields on that hybrid view's business component. For example, on Purchase Orders, to have alerts about changes to the Revision field, activity tracking must be enabled so that the system is 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**.
   When setting up a condition, you can click the toggle icon 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.
   - You can choose to send alerts about changes to fields. See Choosing to Send Alerts about Changes to Fields.
   - 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. You can add a literal user ID or a non-literal user, coming from a business component field. For more information on the **Users** panel, see Alerts View.
   To add a user, click **New**.
   - Literal user ID. In the **User** field, choose a user ID, and then click **OK**. The system displays its user name.
   - Non-literal user. Click the toggle icon, the Users lookup switches to the Fields lookup. In the **User** field, choose a BC field where the user is specified, and then click **OK**. When a non-literal user is selected, the **User Name** column is blank. Note that the system supports users, emails, and employee IDs. You can also add multiple items separated with a colon/semicolon/comma and spaces.

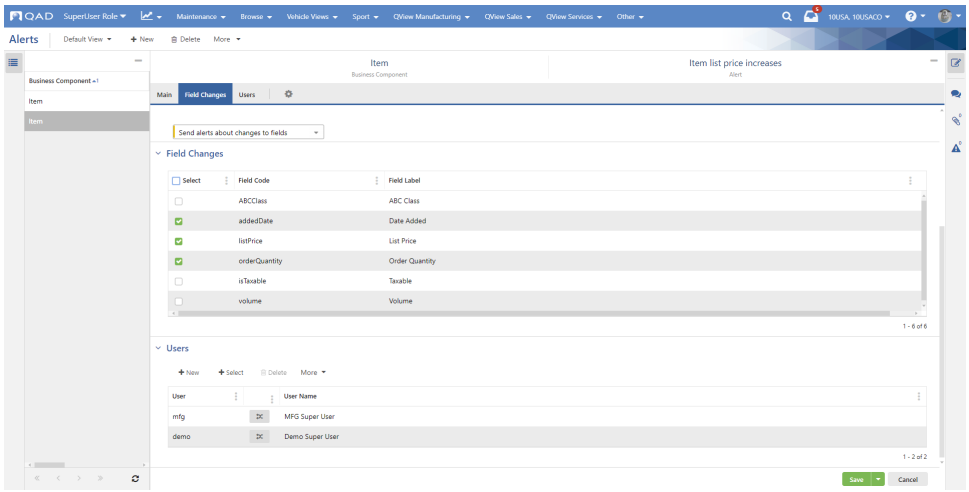


To delete an existing user, choose the record, and then click **Delete**.

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.

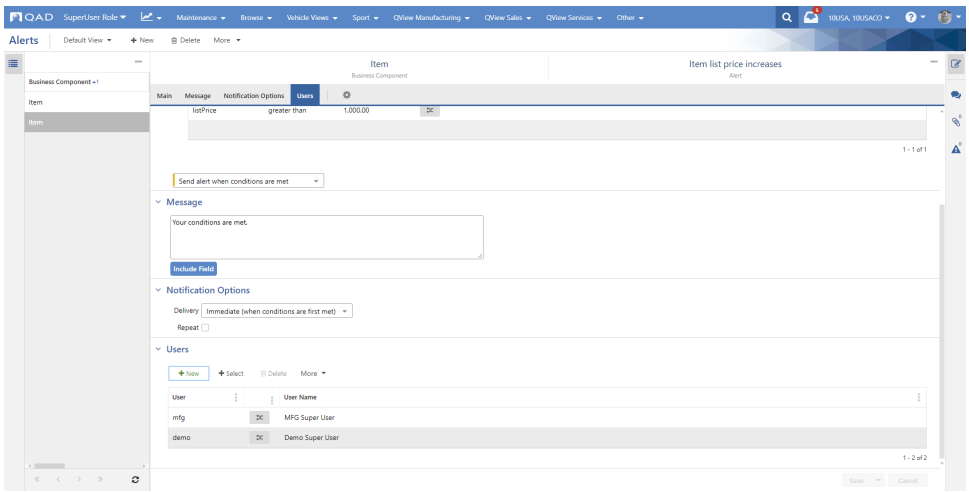


2. On the **Field Changes** panel, select the fields that you want to track. Note that only the fields enabled for activity tracking are displayed here.

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. Extended fields are also available in the list.
   The added fields are automatically added to the end of the text. You can adjust the positions of the added fields manually.

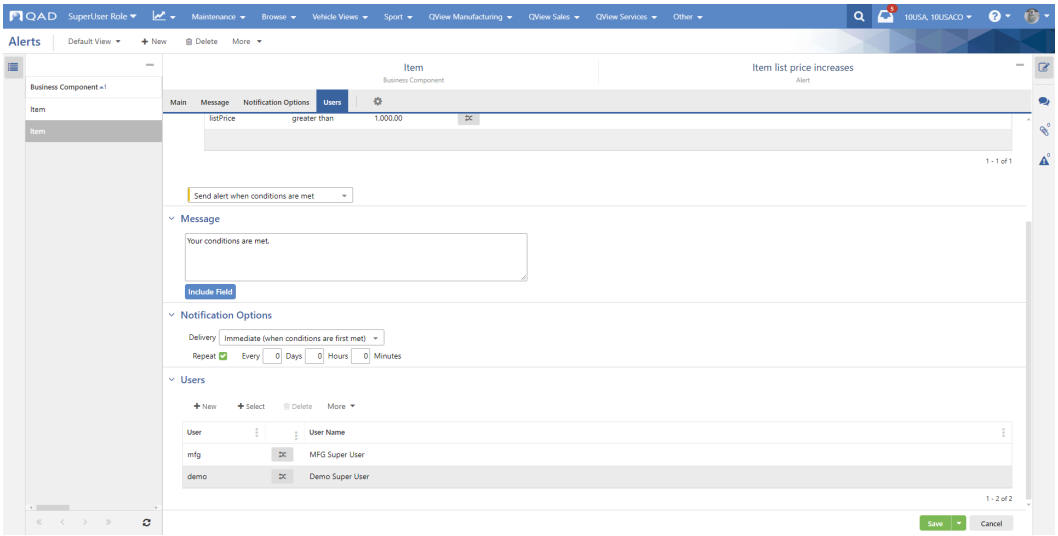


3. On the **Notification Options** panel, configure the **Delivery** setting.
   - **Immediate (when conditions are first met)**: Let the system deliver alert messages when conditions are first met.
     When you choose this delivery option, the first time the conditions are met, the system sends the 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.
   - **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.
   - **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.

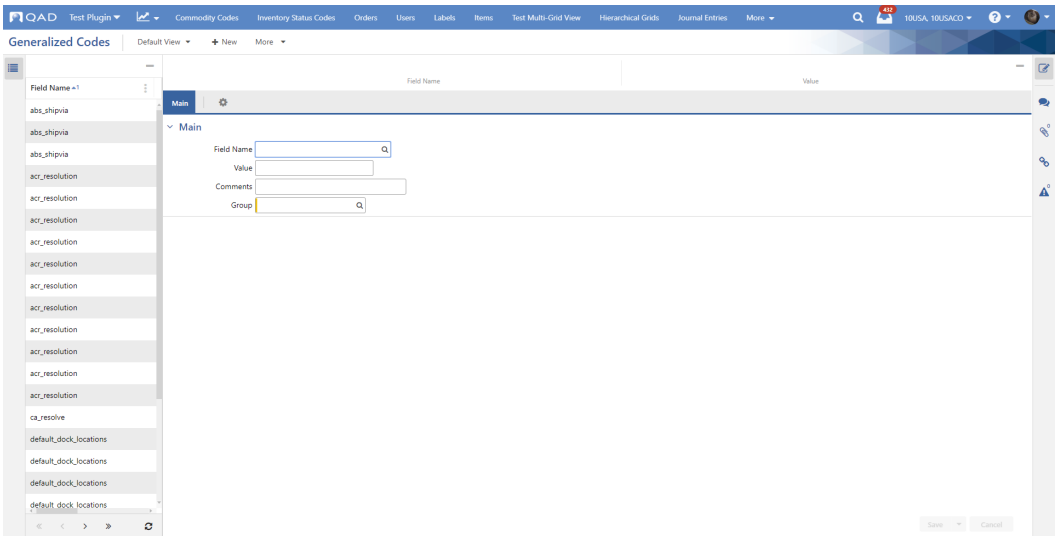
Generalized Codes Configuration

Introduction

Use Generalized Codes to define field values.

When you implement QAD, a number of system and reference fields accept any kind of data, as long as it does not exceed the field length. Use the Generalized Codes view to define values for these kinds of fields.

Many generalized code values are predefined and loaded into the system during installation. Whenever you create a new domain, this default system data is included.



UI Quick Reference

Field Name

Enter the field name.

Value

Enter a valid value. Values cannot exceed the length of the field.

Comments

Enter a comment. The comment displays next to the value in the lookup.

Group

If you have permission, specify the generalized code group that this generalized code belongs to. The group the field is currently assigned to is displayed in this field. This field is only relevant if you are using generalized code groups.

Configuring Drop-Down Lists with Generalized Codes

To create a drop-down list with generalized codes, create a business component with a field of Character data type and not a lookup. In the Physical Field column of this field, enter the name of the generalized code you want to display as a drop-down list. You can use the name of an existing generalized code, such as “stat”, as the Physical Field name, or you can create your own generalized code and use it. If you create your own generalized code, you might need to open System Information > Cache Info, and click the Clear All Caches button.

Generalized Code Groups

Use Generalized Code Groups to define groups of generalized codes for security purposes.

With generalized code groups, you can extend security access to generalized codes beyond the system level down to the job functional level. This enables you to limit the generalized code fields that a user can edit.

Group

Enter a meaningful group name for a generalized code group.

Design Layout Administration

Introduction

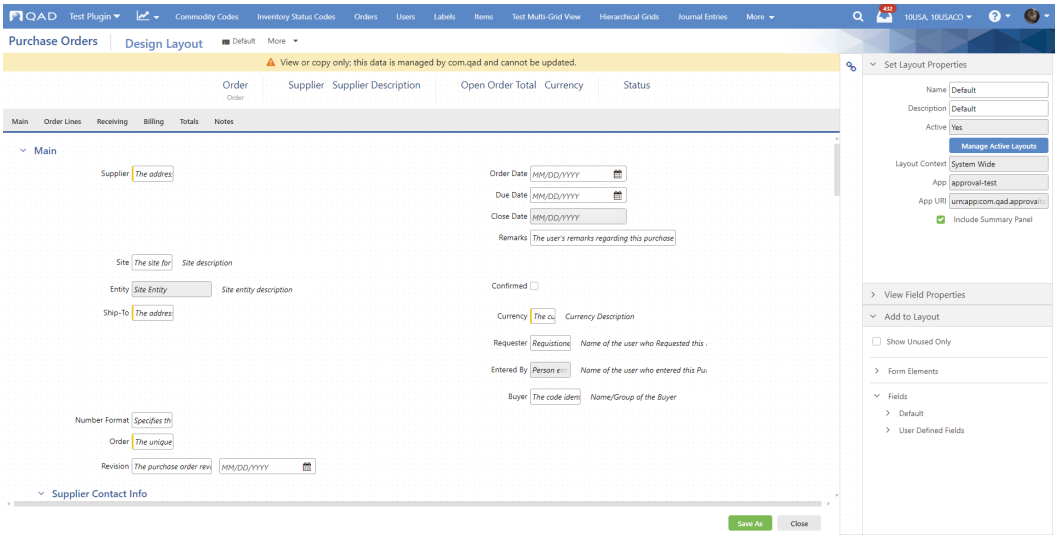
A form organizes everything a user needs for some business task. In a form, a user can create, view, edit, and delete data. In a hybrid browse, which initially displays the full browse, the form opens when the user double-clicks a row (that is, a record) in the browse. A form includes a summary panel, navigation bar, main panel, and then various detail panels and sub-panels.

As an administrator, you can design the layout of a form using the Design Layout feature.

The layouts are based on templates. The Default template is provided by QAD and cannot be changed.

Example Screen Shot

More (on the toolbar) > Design Layout



UI Quick Reference

New Layout

Name

The name of the layout.

Description

A short description of the layout.

Layout Context

Select System Wide for the layout to apply across all domains or select a domain.

App

Indicates the app for the layout.

App URI

Indicates the uniform resource identifier (URI) for the app.

Select Existing Template

Select a template to use for creating a new layout. For example, you can create a new layout from the Default.

Design Layout

In the Design Layout, you can design the form. You can drag-and-drop user interface components, including panels and fields. You can specify panel and field properties, such as their labels and default values.

When you remove the field from the Design Layout, you cannot see it in the Stored view, Design Layout, and Configure Panels. To support the TS Handler code in this case, the IsDesignLayoutHidden attribute is added to the field when it is removed from the Design Layout. So the field is still in HTML, but it is hidden. If the TS code refers to ViewField for the hidden field, it can still access it not causing any errors.

Set Layout Properties

Name

The name of the layout.

Description

A short description of the layout.

Active

Indicates whether the layout is active (Yes or No).

Manage Active Layouts

Click to select which layout is active for the form. Initially, the Default layout is active, but you can change that to any of the layouts that have been defined.

Layout Context

Indicates whether the layout to apply across all domains (System Wide) or to a domain.

App

Indicates the app for this layout.

App URI

Indicates the uniform resource identifier (URI) for the app.

Include Summary Panel

Specifies whether to include the Summary Panel in the layout. This option is available only if the Include Summary Panel checkbox is selected in the Form Builder.

Configure Drill Downs

To access the Configure Drill Downs panel, click the link icon located on the left side of the Set Layout Properties panel. With the Configure Drill Downs panel, you can identify the drill downs that do not appear for linked grids and actions and re-define those drill downs. The warning icon appears next to the drill downs that are not visible. These are the browse-based drill downs: .Net UI - Browse Based and User Added - Browse Based. You can view the drill down details by clicking the pencil icon next to the drill down. The drill down type is available on the Summary Panel in the Drill Downs detail pop-up window. For more information about drill down types, see Form Builder - Views - Drill Downs.

Note: To add a new drill down, click the plus sign on the Configure Drill Downs panel.

View Field Properties

Edit

Click **Edit** to change the field properties for the business component. See **Business Component Field Properties** below.

Field

The field name.

Field Label

The field label.

Business Component

The business component for the field.

Detail Table

The detail table for the field.

Physical Field

The physical field table identifier.

Field Length

Indicates the display length of the field.

Format

Indicates the format for the field. For example, a field whose value is twenty characters is indicated by x(20).

Default Value

Enter the default value for the field.

Required

Select whether the field is a required field, requiring a value from the user.

Business Component Field Properties

Field

The field name.

Business Component

The business component for the field.

Detail Table

The detail table for the field.

Physical Field

The physical field table identifier.

Layout Context

Indicates whether the layout to apply across all domains (System Wide) or to a domain.

Default Value

Enter the default value for the field.

Note: Changes to this property affect all layouts using this business component.

Required

Select whether the field is a required field, requiring a value from the user.

Note: Changes to this property affect all layouts using this business component.

Field Label

Use the Labels lookup to select the label for the field.

Note: Changes to this property affect all layouts using this business component.

Field Length

Indicates the display length of the field.

Note: Changes to this property affect all layouts using this business component.

Format

Indicates the format for the field. For example, a field whose value is twenty characters is indicated by x(20).

Note: Changes to this property affect all layouts using this business component.

Reset

Reset field properties back to their original values.

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 **Form 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.

Form Elements include Panels. Under Form Elements, you can select an element and then drag it to the form area, placing it where you would like the element on the form.

Fields include all the fields available, organized by panel. Under Fields, you can select a field and then drag it to the form area, placing it where you would like it on the form. Fields must be located within a panel. Fields listed with a checkmark are already included on the form. Fields listed with a plus (+) sign are not currently included on the form. Fields with a yellow horizontal indicator are required fields.

Exporting and Importing Design Layout Data

Introduction

The following topics describe how data created using the Design Layout tool can be exported from one environment and imported into another environment.

Exporting Design Layout Data

1. View layout and business component property data created using the Design Layout tool is written to the application database and is associated with the module URI urn:module:customization.
2. All data associated with this module URI can be exported by executing the following two YAB commands:

    yab metadata-export urn:module:customization -type:layout -dir:/directory_path

    yab metadata-export urn:module:customization -type:entityfieldoverride -dir:/directory_path

This will create the following structure in the /directory_path directory:

    |--entityfieldoverride

        <entity field override xml files>

    |--layout

        <view layout xml files>

Creating a Module Package

The YAB command system-package-create can be used to create a package for the data exported above. In the following example, a package named cust001 with moduleURI=urn:module:customization.001 is created that includes all the files in the current working directory.

    yab -class:cust001 -attribute:module.uri=urn:module:customization.001 -version:1 system-package-create .

This will create a package named cust001-1.0.0.0 in the location where the YAB command is executed.

Loading a Module Package

The module package created above can be loaded into another environment using the following commands:

    yab install cust001-1.0.0.0.zip -install-update:false

    yab info (should now show the cust001 package)

    yab metadata-cust001-update

Analytics

This section provides training materials about QAD Enterprise Platform Analytics.

Please see the online help for Action Centers: QAD Online Help - Action Centers (March 2020).

This section covers the following topics:

- Platform Analytics Introduction
- Action Centers Overview
- Create Action Center and Use Action Center
- Create a KPI
- Create Visuals in the Thinkspace
- 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 a previous version of Action Centers. The overall functionality remains the same but screens might slightly differ with the latest 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.

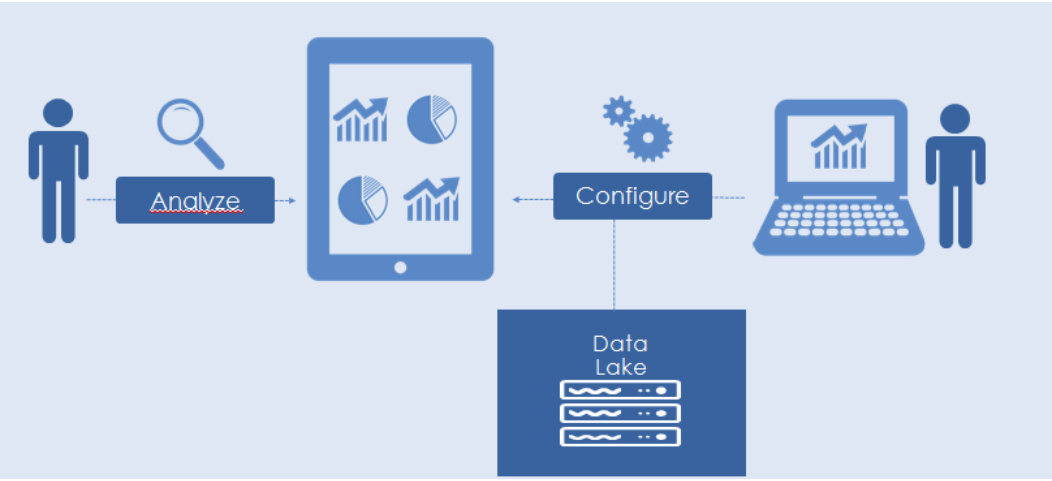
The main UX are the Action Center dashboards containing visual Key Performance Indicators (KPI).

From the Action Center, the users can further analyze 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

Main features

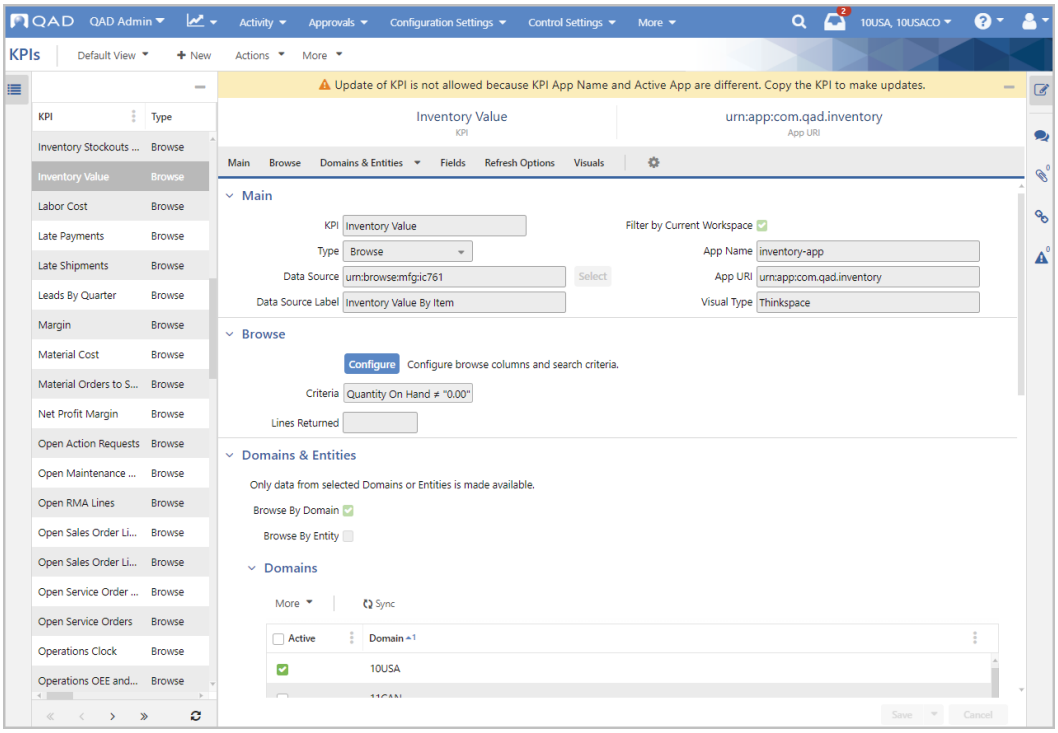
Users can create Action Centers and populate those with the KPIs of their choice.

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:

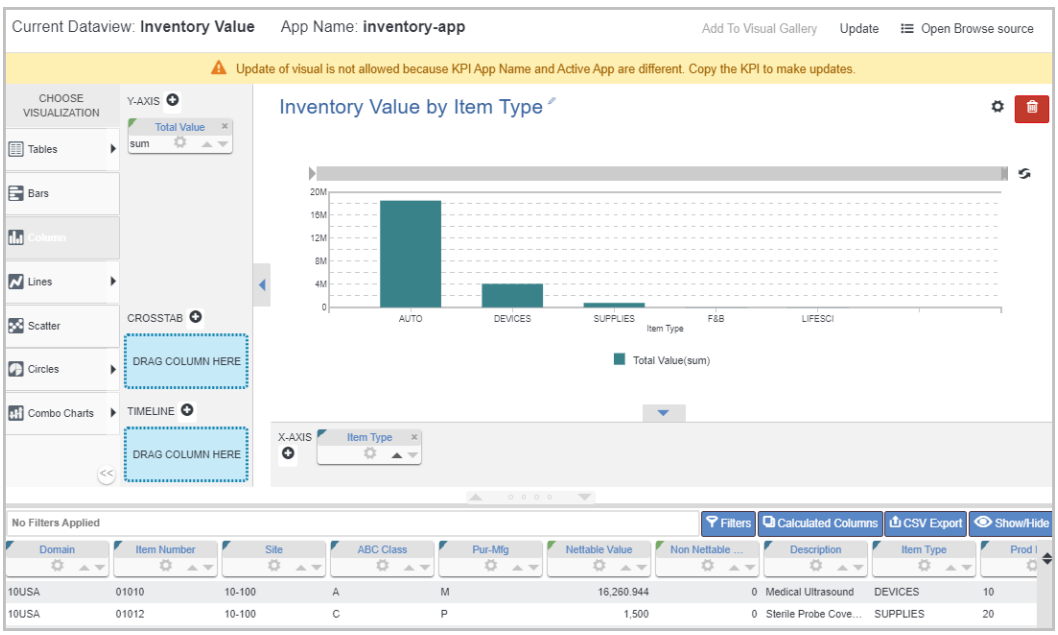
KPI maintenance

Use the KPIs menu to define the KPI. This defines the data source (browse) and the data selection (filter, domains, columns) for the KPI.



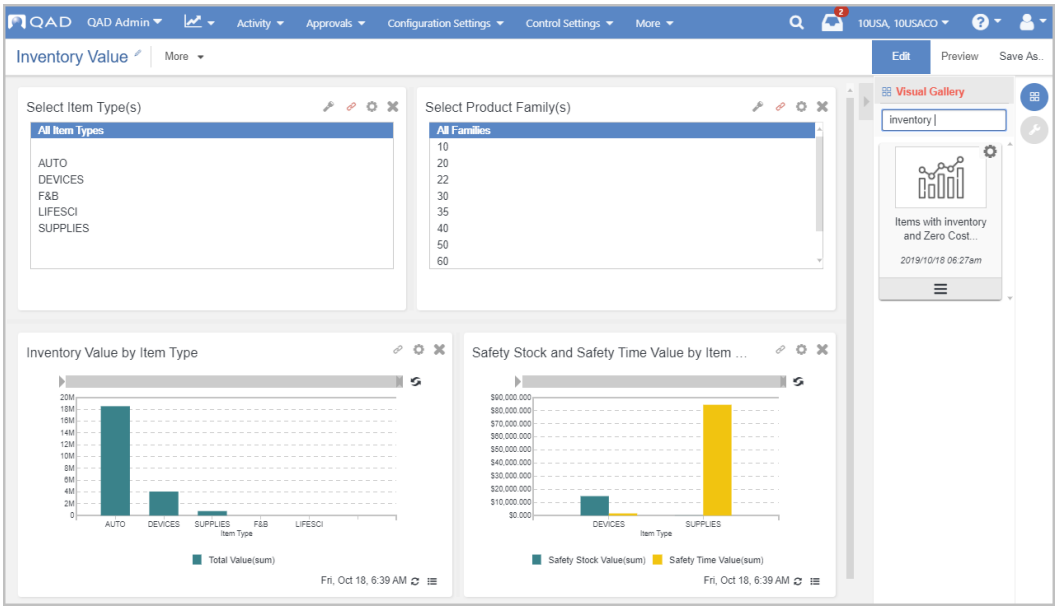
Visuals

From the KPI screen you can proceed to the visual creation (charts, gauges, tables) with calculated columns and filters where needed.



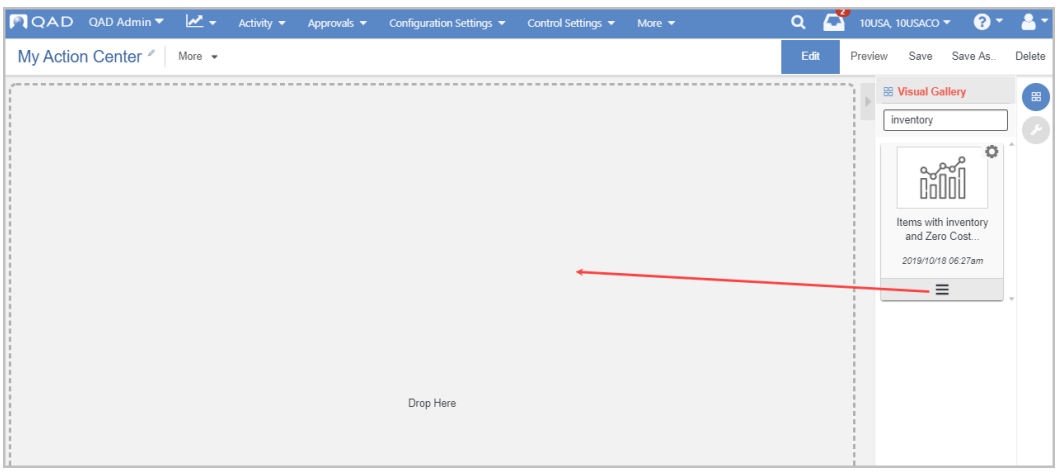
Publish

Visuals are then made available for Action Centers by publishing them in a shared Gallery.

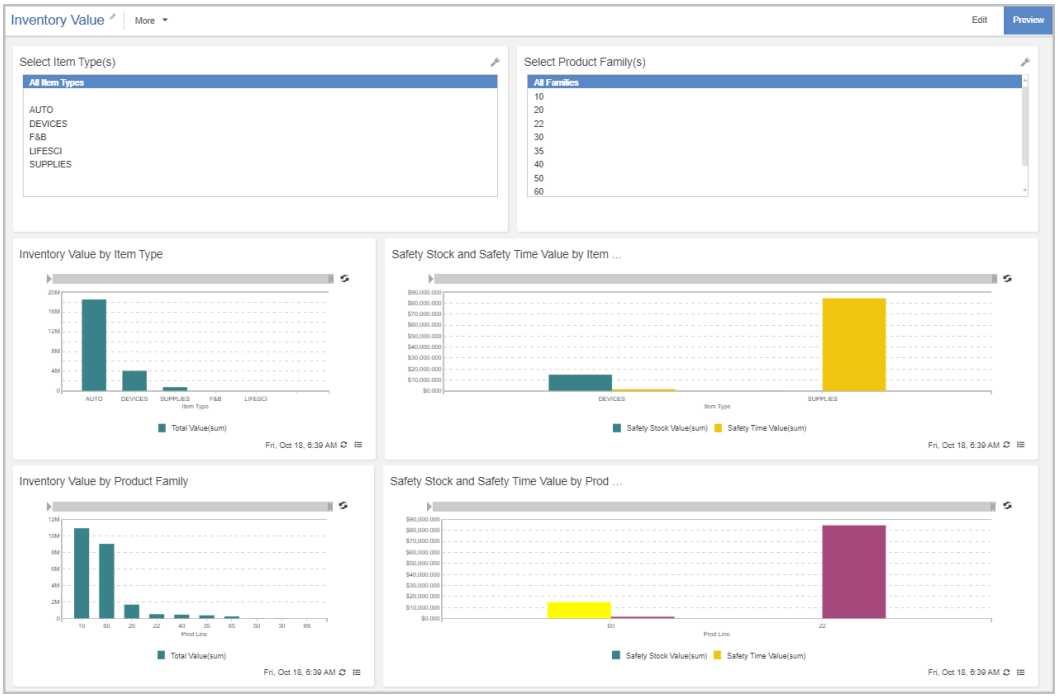


Use in Action Center

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.


Inventory Value
More
Edit
Preview
Select Item Type(s)
All Item Types
AUTO
DEVICES
F&B
LIFESCI
SUPPLIES
Select Product Family(s)
All Families
10
20
22
30
35
40
50
60
Inventory Value by Item Type
AUTO
DEVICES
SUPPLIES
F&B
LIFESCI
Item Type
Total Value(sum)
Fri, Oct 18, 6:39 AM
Safety Stock and Safety Time Value by Item ...
DEVICES
SUPPLIES
Item Type
Safety Stock Value(sum)
Safety Time Value(sum)
Fri, Oct 18, 6:39 AM
Inventory Value by Product Family
Prod Line
Total Value(sum)
Fri, Oct 18, 6:39 AM
Safety Stock and Safety Time Value by Prod ...
Prod Line
Safety Stock Value(sum)
Safety Time Value(sum)
Fri, Oct 18, 6:39 AM

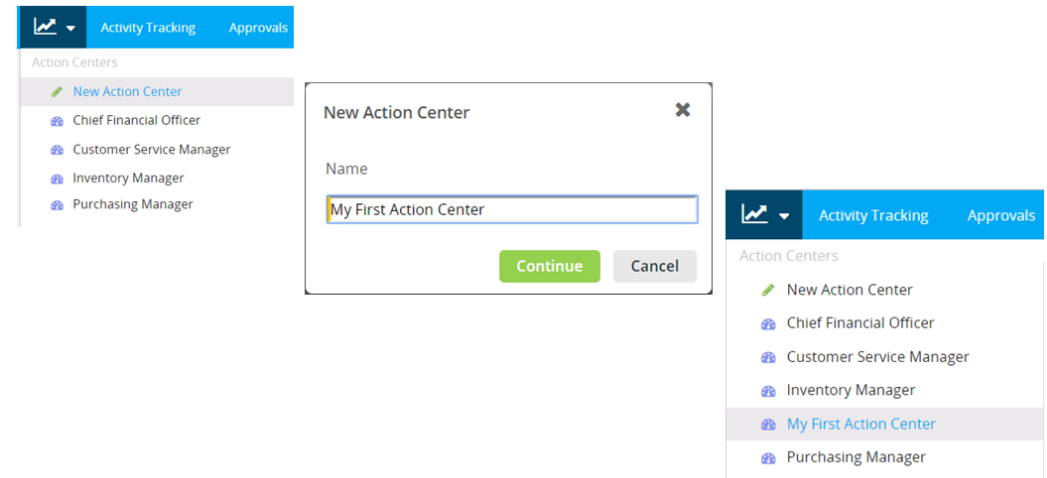
Create Action Center and Use Action Center

Create a 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.



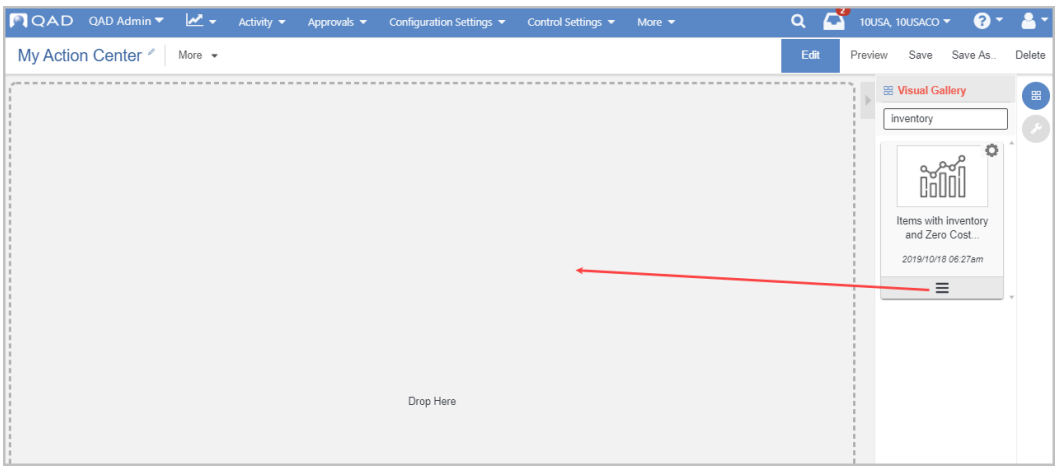
Select KPI Visuals from Gallery

When you open the new action center, it is still empty. Click Edit 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.

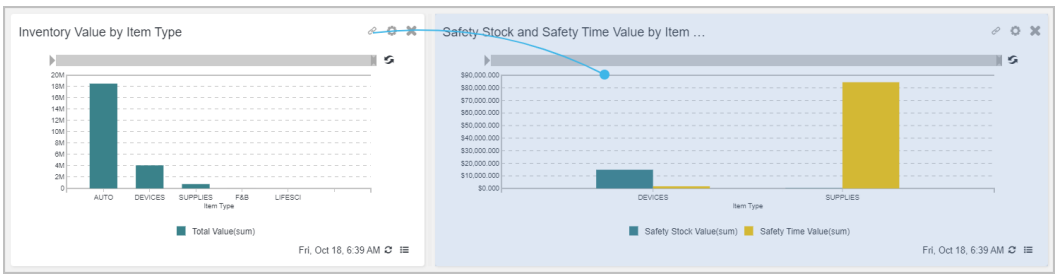
Now all KPIs that have Inventory in their name will show.

You can drag and drop the KPIs that you want from the gallery to the action center, arranging them as you like.

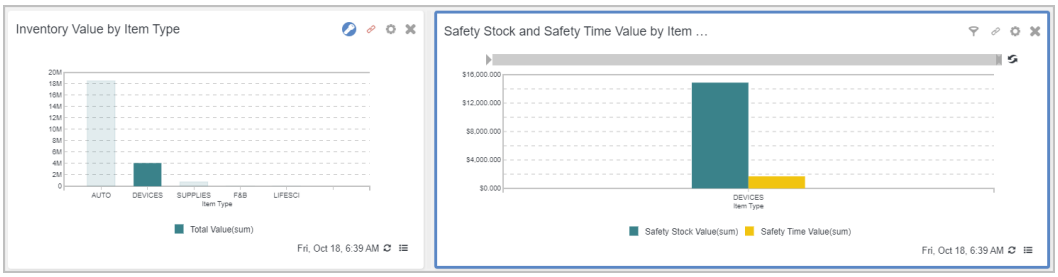


You can also add links between visuals in the action center. Selecting data points in the source visual updates the other visuals to display only the data that relates to the selected content in the source visual. Drag the link icon from the source visual and drop it on one of the target visuals.

Permitted target visuals display shaded in blue.

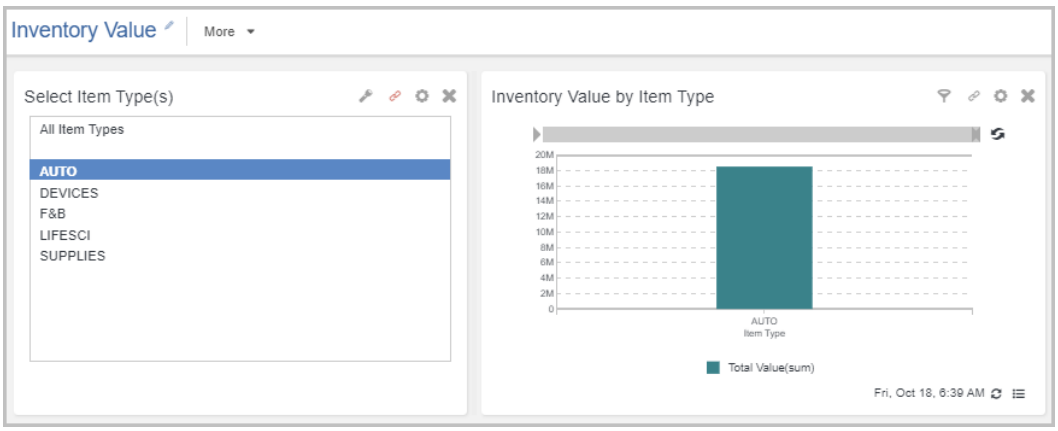


The link icon in the source visual shows in red.



Note: Each visual that you want to link must be based on the same KPI definition (dataview) from the KPIs View, so that you can filter on a field within the dataview, or based on dataviews that share a Thinkspace Field Name in the KPI definition.

You can also use the linking feature, in conjunction with list widgets and date/time widgets, to filter the contents of action center visuals.



Analyzing a KPI Visual

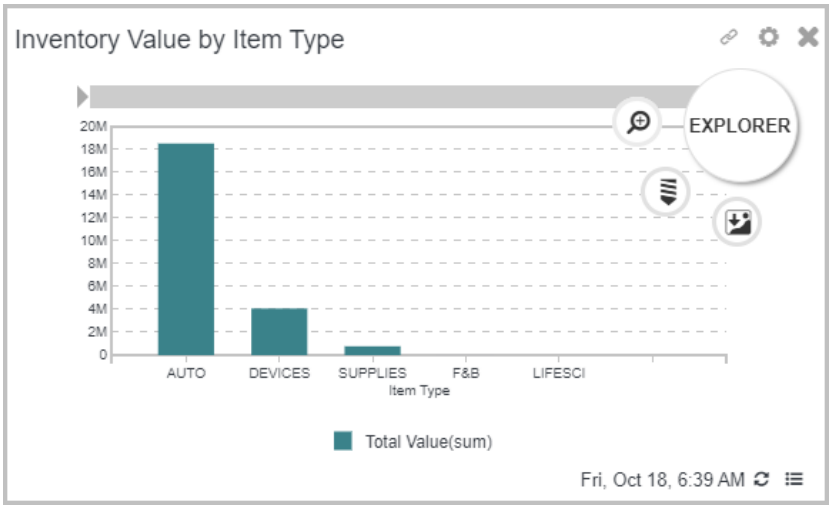
You can analyze the KPIs in the action center by hovering over the graphs to see the values of the data points.

You can also click on the legend to show and hide the details of the chart.

Clicking the link icon allows you to check if the visual is linked to any other visuals in the action center.

The Explorer menu is visible when you mouseover the area above and to the left of the visual, below the configuration icon. It offers different modes to analyze the visual data. Zoom mode provides a ruler above the visual. You can drag this ruler to zoom in on the visual details. Select mode is only available when the visual is a source widget for linking to other visuals. Drill mode allows you to drill down to the details of a particular dimension in the visuals. When you select a dimension to drill down to, these details display below the visual title. A filter icon appears next to the link icon.

Click this link to see the exact drill-to details.

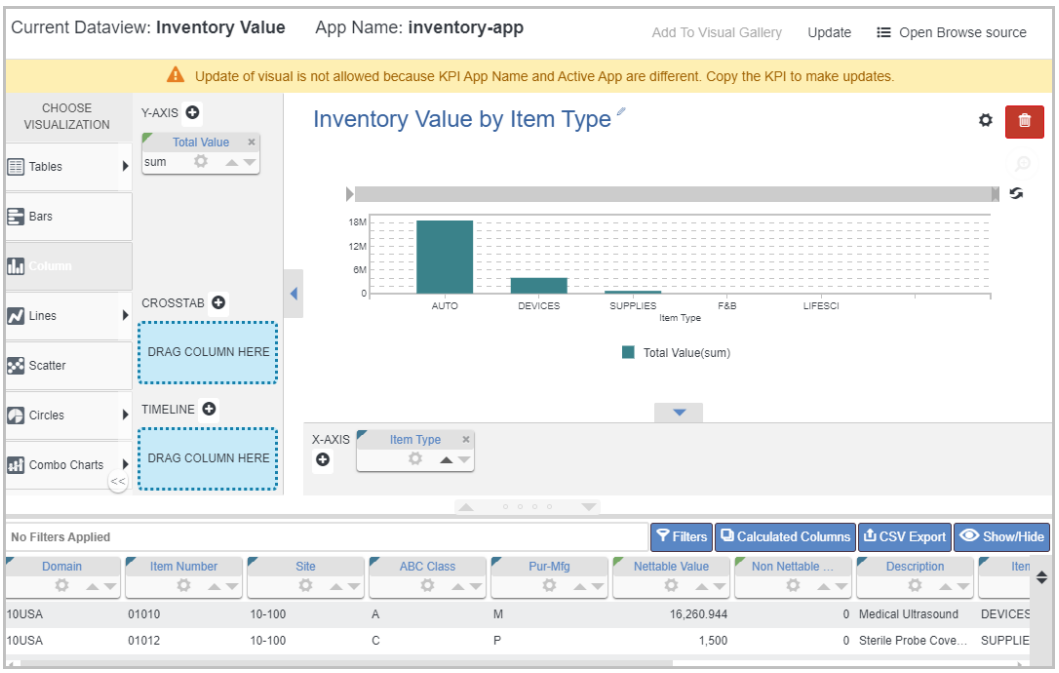


The gear or configuration icon allows you to edit or rename visuals. It is also useful if you want to learn more about the current configuration of the visual.

Click Edit in the upper right corner of the action center. Then, click the configuration or gear icon for the visual and click Edit Widget to open the visual in the Thinkspace.

Here you can fine tune the visual. After previewing the chart, you can decide to update the existing visual in the action center, or add the visual to the Visual Gallery.

Note: The Current Dataview field in Thinkspace matches the name of the KPI definition in the KPIs view. The App field displays the name of the app in which the visual was created.



Refreshing an Action Center

You can refresh all of the visuals in an action center at once.

From the More menu, to the right of the action center title, click Refresh.

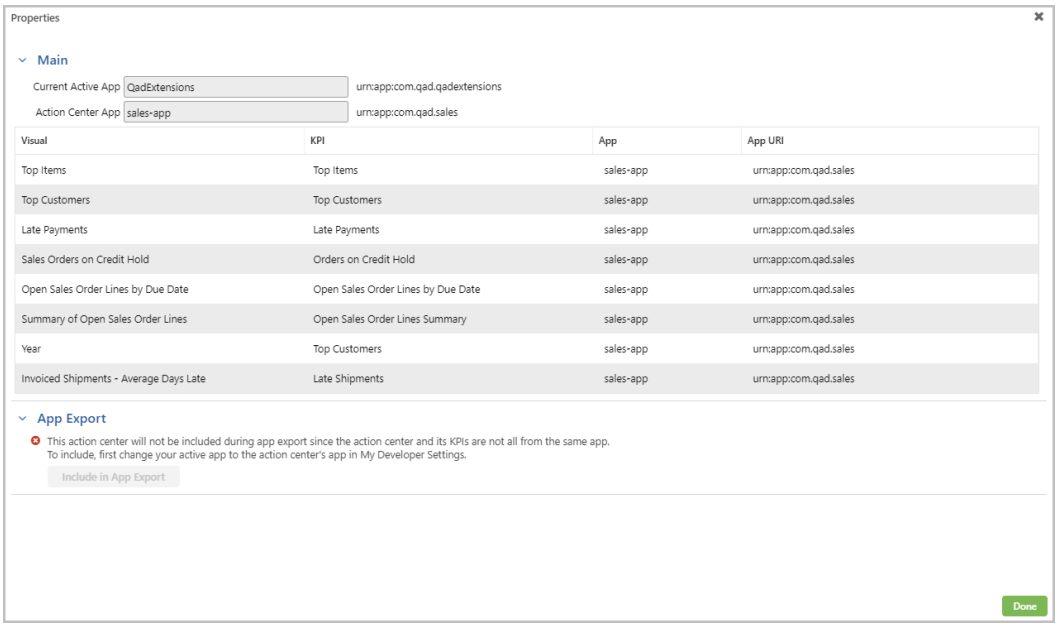
A Processing message appears on the screen while the visuals are refreshing.

After the visuals have refreshed, they display the updated contents. The refresh date at the bottom right of the visual is updated to the current date and time.

Viewing Action Center Properties and Export Options

You can use the Properties screen to view details of an action center, and to check if the app is ready for export through the yab app-export procedure.

From the More menu, to the right of the action center title, click Properties to view details of each visual, along with the name of the underlying KPI definition and associated app. The name of your current app is also displayed.

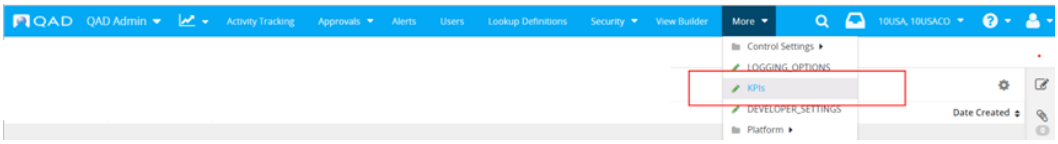


To check if this action center will be included in the action center app if you export that app, scroll to the App Export section. If the action center contains visuals that are stored in different apps, then you must change your active app to the action center app. If you make this change, the Include in App Export button is enabled. If you click Include in App Export, the KPIs and visuals that reside in other apps are copied to the action center app—which is now your active app. You are then free to complete the yab app-export command to export the app and the action center that it now contains.

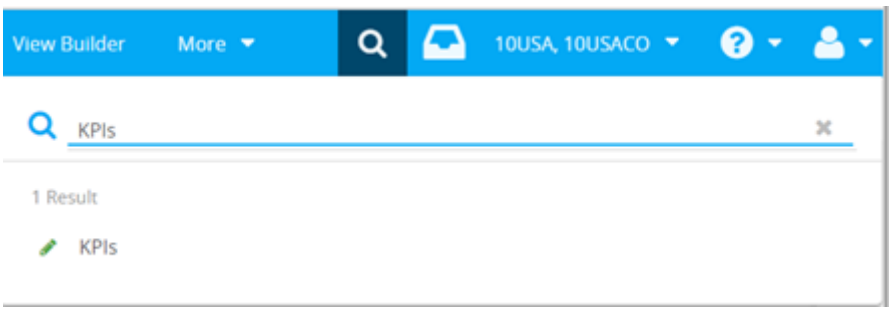
Create a KPI

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.



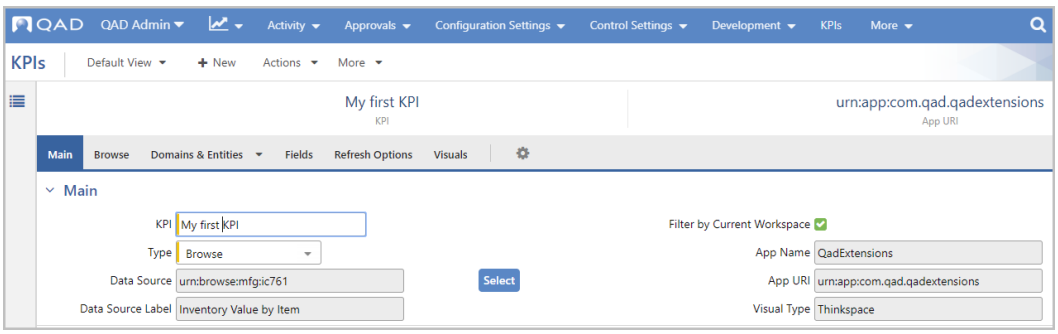
Even if you don't have the QAD Admin menu can still search the KPI maintenance by typing KPIs in the menu Search.



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 the Main panel you enter a meaningful name for the new KPI.

The Type field indicates the type of data source that will be used for the KPI, and is currently read only.

The App Name field shows the app in which the KPI data is stored. This field is read only, and is set to the app selected as your active app in the My Developer Settings screen.

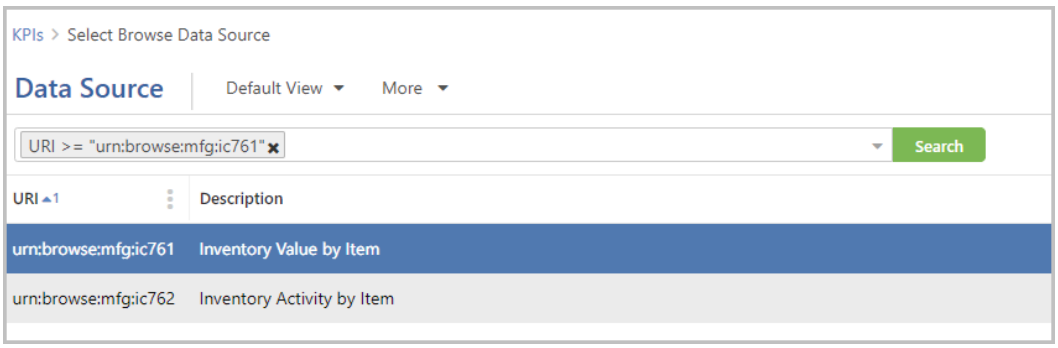
Selecting a browse

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 “Inventory Value by Item”. This returns the ‘Inventory Value by Item’ browse.

If you know the browse code, for example “ic761”, then you can search it with "URI Contains ic761". This also returns the ‘Inventory Value by Item’ browse.



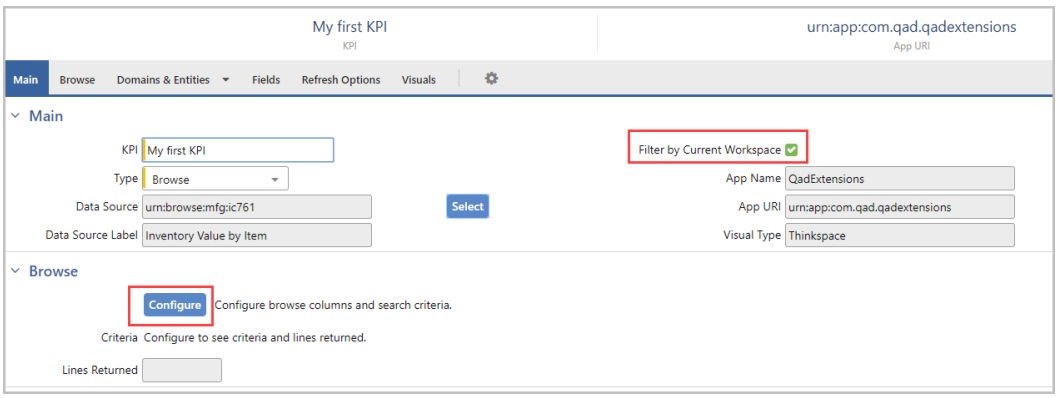
Once you located the browse, you select it and click OK.

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.

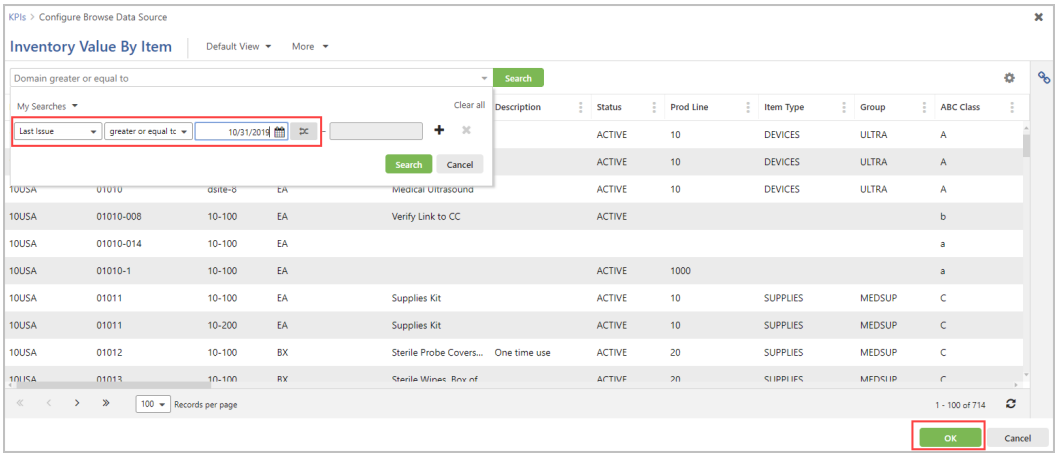


Next optional step is to configure the browse columns and search criteria. Search criteria are important to limit the number of records that is returned from the browse.

Therefore you click the Configure button. This opens the actual browse that you selected earlier.

In this example the Inventory Value by Item 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.

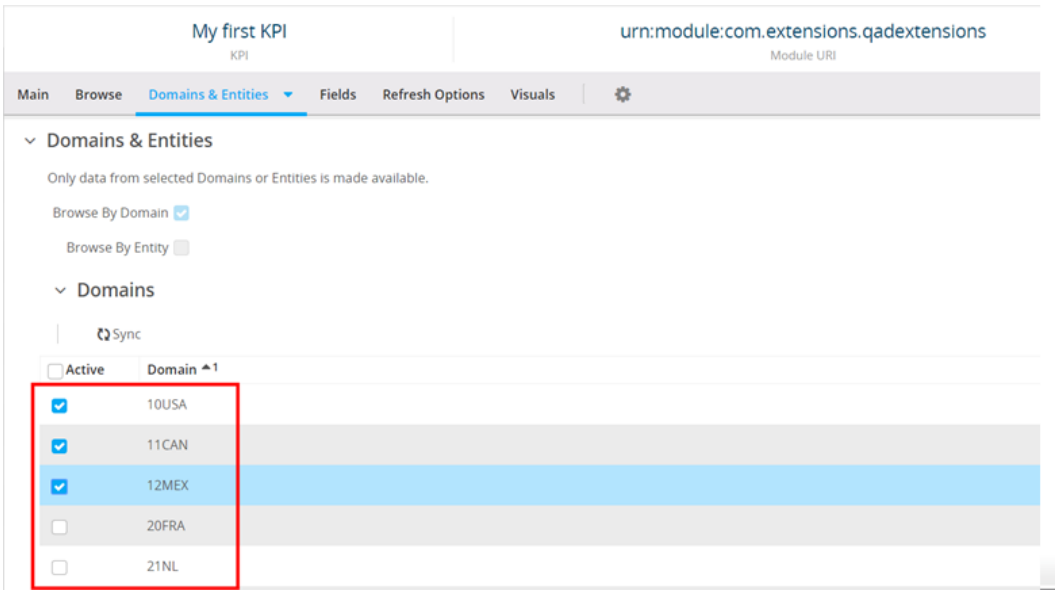
You can always change the search criteria later.

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.

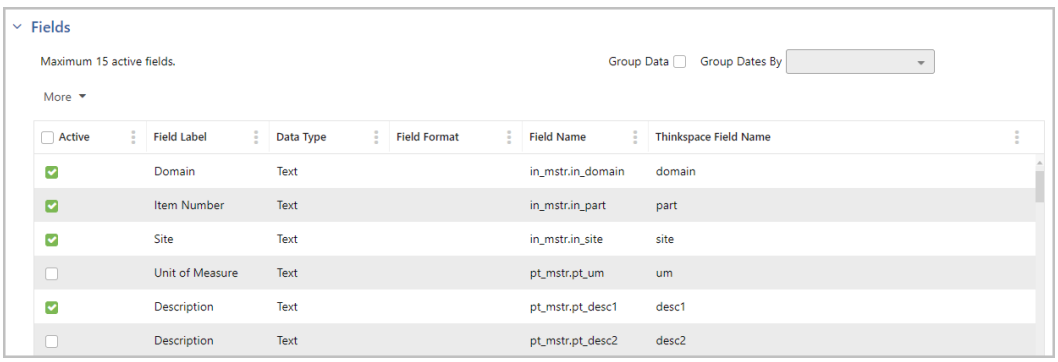
In the grid, you can also change the field labels if that is needed. For Date and Number type fields you can also set the Field Format. That is a display mask used for the fields in the visuals.

Important: You can add a field to the list of active fields, or modify the Field Label or Field Format attributes of an active field, at any time. The data saved and displayed in Visuals updates to reflect this change. Note that this Visuals update can take some time to process. You can also remove a field from the list of active fields at any time. Any formula, filter, or unpublished visual that references the field is deleted. Published visuals that reference an inactive or modified field are still visible in action centers. However, they may not display the expected data because a field that is used in the visual is no longer accessible or has changed.

Data Type and the original browse Field Name columns are read-only and for information purpose only.

The Thinkspace Field Name allows you to identify fields in different KPI definitions as containing the same type of data. For example, a field in KPI definition A has a Field Label value of Order, while a field in KPI definition B has a Field Label value of Order Number, but both of these fields contain the same type of data. Identifying commonalities between KPI definitions allows you to link visuals, based on different KPI definitions, in an action center. The Thinkspace Field Name is the basis for these links. It is not possible to update the Thinkspace field after you save the KPI definition. For this reason, it is important to think about which visuals you want to link in an action center, and to set the Thinkspace Field Name values in the related KPI definitions at the outset.

The Group Data option allows you to group the browse records based on the active fields. This grouping is performed for all active text and date fields. Active number fields are summed automatically. Grouping allows you to improve performance by reducing the granularity of browse query results. For example, a transaction history browse with unnecessary information about each item movement could be grouped to provide a daily or weekly total of item movements, if this level of detail is sufficient. After you select the Group Data option, you can specify how you want to group date fields—when applicable—using the Group Dates by option. The results of the grouping are visible in the Thinkspace, which is accessible from the Visuals button at the bottom of the screen. While grouping is primarily used to improve performance, you can also use it to perform calculations over multiple rows. For example, to calculate the percentage margin for sales in a particular entity or month, you must first calculate total margin and total sales values. The formulas in Thinkspace cannot calculate totals over multiple records. Grouping the records by entity or month to create total margin and total sales values allows you to calculate the percentage margin.

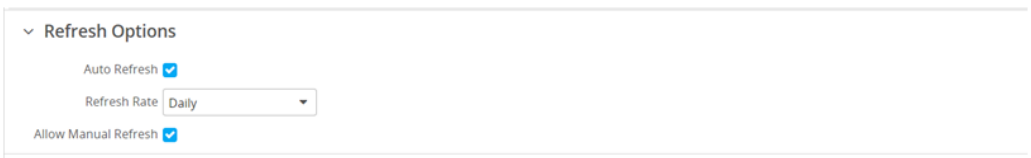


Refresh Options

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.

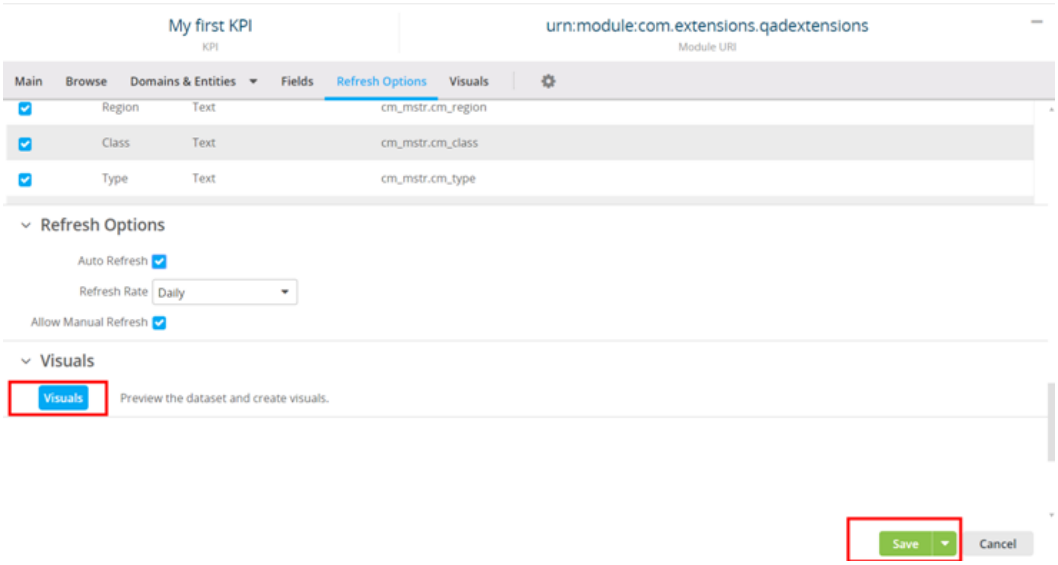


Data refresh is discussed in more detail further on.

Save and Visuals button

Now we have completed the KPI definition and we can save it by clicking the Save button.

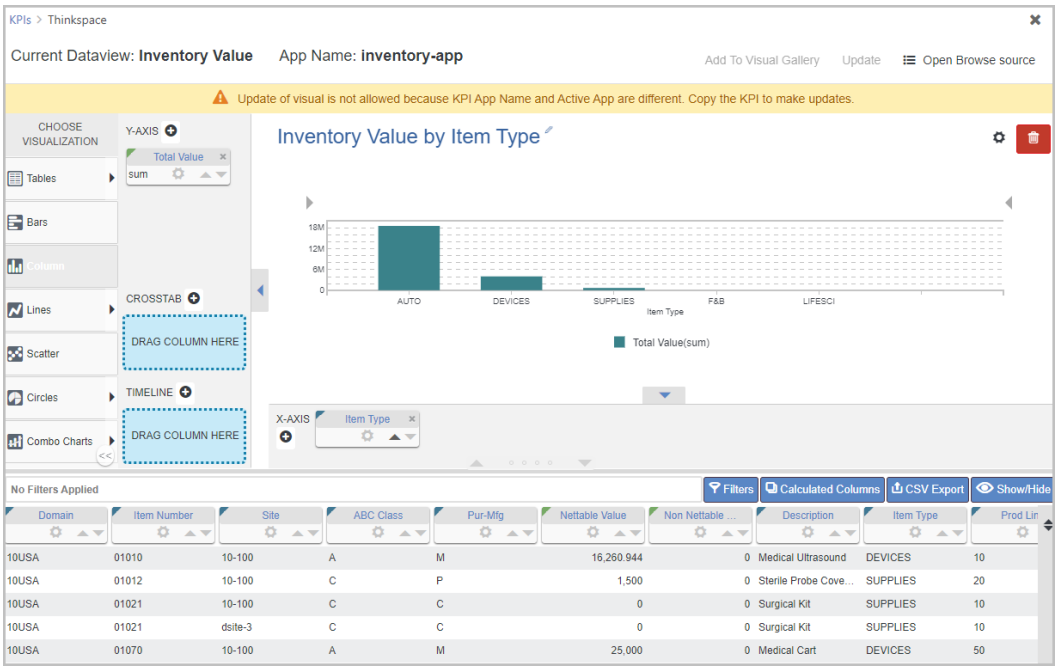
This also enables the blue Visuals button.



Visuals Preview

Clinking the Visuals button brings you to the Thinkspace, where you create the graphical representations of the KPI: charts, gauges, tables, etc. The creation of visuals is explained on another page.

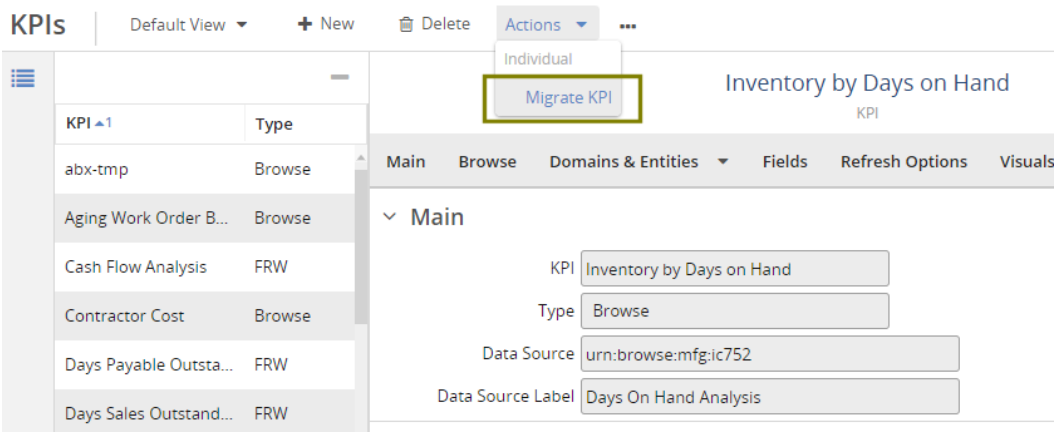
Note that there is always a table available on this screen that shows the data returned from the KPI data source (in this case the Inventory by Item Value browse).



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 in the Thinkspace

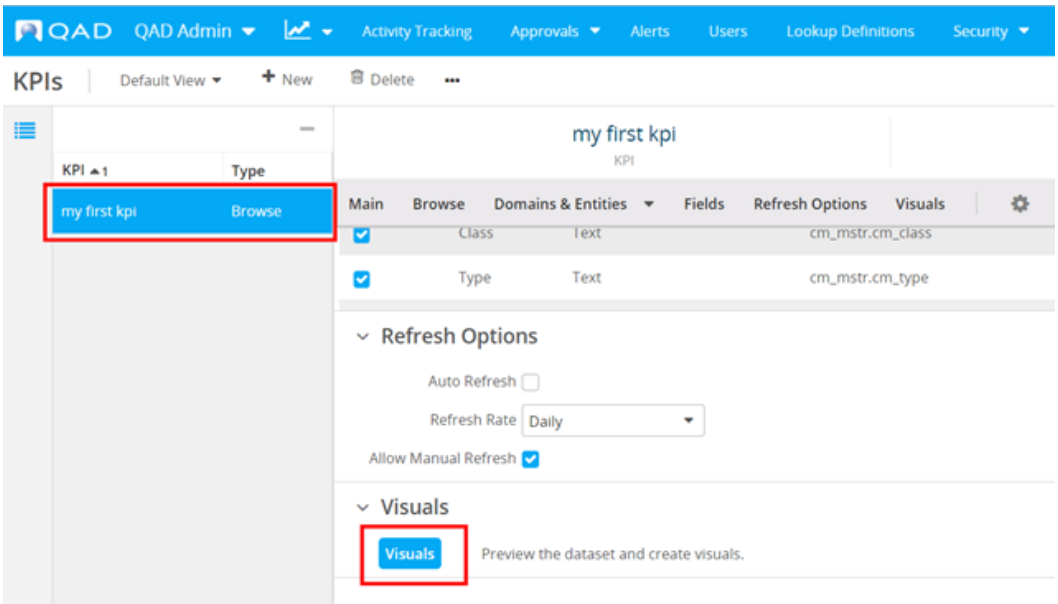
Accessing the Visuals page

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.

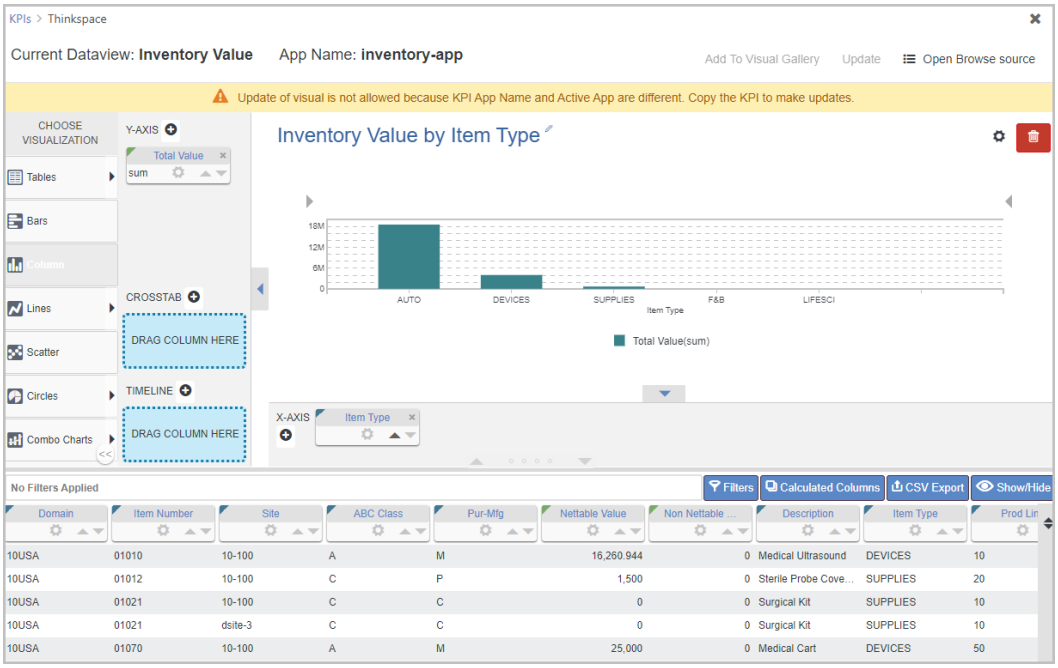
In the first example, we start from a KPI in the KPI maintenance screen. On another page, it was explained how you create a KPI in that maintenance screen.

Now we select a KPI from the browse and go straight to the Visuals button at the bottom of the screen.



Clicking the Visuals button takes you to the Thinkspace, and displays the visual that was last saved to the gallery from this KPI definition, when applicable.

Note: You can also access the Thinkspace from an action center by clicking Edit in the top right corner, clicking the Configuration icon on the visual, and clicking Edit Widget.



Visuals Toolbar section

This section identifies the KPI definition and app associated with the visual. It also allows you to update a visual or add a visual to the Gallery, and to access the source browse on
which the associated KPI definition is based.

Visual settings are saved in two locations—the Gallery, and in each of the action centers in which the visual is used. Each time that you add a visual to the Gallery using the Add to Visual Gallery option, it is saved as a new visual in the Gallery, with the user name of the creator and a time stamp.

When you access the Thinkspace from the KPI definition, the Update option allows you to update the current visual in the Gallery. This option is only available when the visual has already been added to the Gallery. When you access the Thinkspace from an action center, this option allows you to update the current visual in the action center, only, with no impact on the Gallery.

Open Browse Source allows you to view the browse results. All filters added using the Configure button in the KPIs View are applied to the browse results in this view. Most Thinkspace filters are also applied to the browse results, but filters that use the "not in list" condition, filters that include SQL functions, and filters with parenthesized combinations of AND and OR operations are not applied to the browse data in this view. Common functions, such as TODAY() and THISYEARSTART(), are applied. If filters are not included, a message appears to warn you that not all of the filter criteria can be applied to the browse results.

Note: A KPI definition can include multiple domains or multiple entities. But when you run the browse in the Web UI, then the Current Workspace setting is applied to it and the browse only shows records from the current domain or entity.

Visualization Menu

This menu on the left allows you to select the type of visual that you want to create, and to switch between visual types—for example, to switch from a bar chart to a pie chart. It contains a list of the available visuals.

Visualization Area

This is the canvas that you can use to create and display visuals. The first time that you open the Thinkspace for a particular KPI definition, this area is empty. Using the other areas of the Thinkspace screen, you can create a visual. To clear the canvas, click the trash can icon on the top right on the area.

Data Table Area

The data table at the bottom of the screen displays the records generated from the KPI configuration. When the browse results are grouped using the Group Data option for the KPI definition, a Record Count column appears in the data table to indicate the number of rows within a group.

Data Enrichment Area

The options available include:

• Filters. The filter button allows you to filter the contents of the table based on a particular column. You can build complex filters by adding and connecting simple filters. Click Filters and, following the on-screen instructions, drag a column pill or header into the Filters area. When prompted, specify the values that you want to apply as a filter to that column and click Apply. You can continue to drag column pills into the filter area to build up a complex filter. You can group filters, or apply AND and OR operations to them. You can delete individual elements of a complex filter, and switch filters on and off at the top level. Any filters that you add appear to the left of the Filters button.

• Calculated Columns. This option allows you to add a column to the data table. The content of each cell in this table column is the result of a specified calculation or formula. The expression syntax must conform to the SQL syntax rules for PostgreSQL. For more information, see the PostgreSQL Reference Manual. Calculated columns allow you to create custom time buckets for reporting purposes, compare dates to identify overdue orders or payments, round numbers, convert data from one type to another, or manipulate text strings.

• CSV Export.

Further Information

For detailed explanations of how to use the Thinkspace to create visuals, see the following documents, which are taken from Logi Analytics help content at logianalytics.com:

Using the Thinkspace

Working with Charts

Working with Columns

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

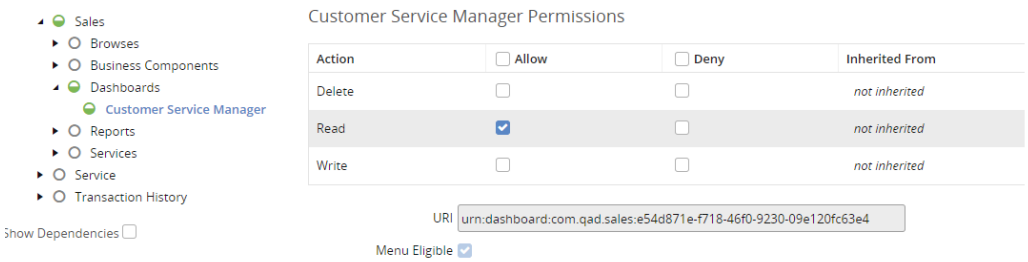
The second type of users are the administrators and power users who can create KPIs, can create KPI visuals, can publish KPI visuals to the shared Gallery and they can create shared role based action centers. For all these objects they also have rights to update them or even delete them.

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”



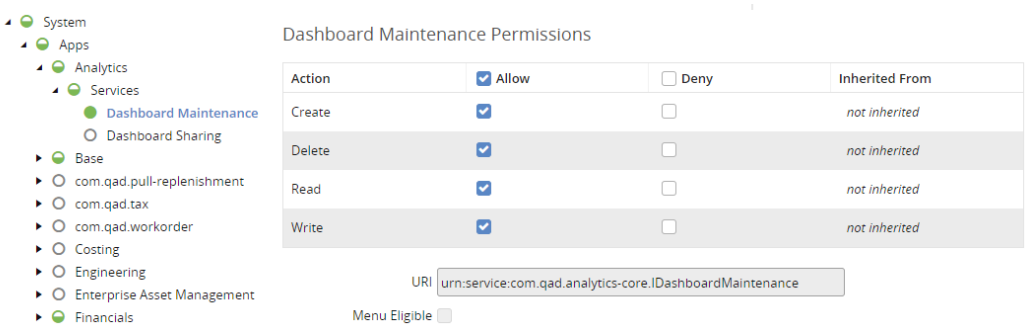
Read access means you can see and access the action center, analyze the KPIs but you cannot add/remove KPIs from the specific action center.

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 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

Record-Level Security for Browses Created with the Platform Business Component Builder

You can control access to browses created using the platform component builder at the individual record level. This level of security is switched off by default. Record-Level Permissions are managed by your administrator, and relate to data access in general—not specifically to Action Centers. For more information, see the Security section of the online help.

When a platform business component has record-level security applied, this will also automatically filter the data that is visible in action centers.

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.

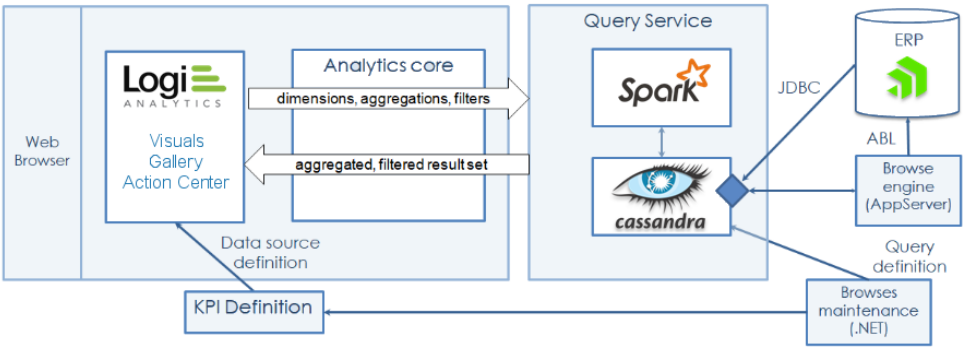
In earlier versions of Action Centers, we set a limitation of 5,000 rows retrieved by the browse to avoid performance bottle necks in the rendering of the KPI visuals.

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. Therefore 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 at 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

Platform Scripting - TypeScript

Introduction

In Platform, there are two places where an app can be extended with scripts, namely on the UI and on the server. The scripts can be developed in TypeScript, at runtime it is JavaScript scripts running in the browser or on the server.

The two types of scripts use the same programming language, but they have a different API. Client scripts are scripts acting on UI events, while server scripts are business logic extensions. This page describes in short the two types of scripts. More detailed information and a guide on how to develop them can be found on the child pages.

Client-side scripts

Client scripts are running on the client, in the browser. Client scripts can only act on UI events of the maintenance view, not the browse part of a hybrid view. A client script can act on different parts of the maintenance screen by attaching a certain type of event handler to the screen. These different handler types can be created inside the script. The following different types can be created:

- A main event handler that handles general events of the maintenance screen itself. Examples are:
  - Initializing, saving, and closing the screen.
  - Clicking an action button or a button on the bottom button bar (close, cancel, etc.)
  - Changing the record, creating a new record, refreshing the data in the screen, and so on.
- A form event handler. It can handle all the events that happen in the form on the screen. The form is all the fields and buttons between the top menu bar and the bottom button bar. Examples are:
  - Modifying a field value
  - Clicking a form button
  - Act on field leave
- A grid event handler. This is an object that handles grid events such as:
  - Modifying a grid cell
  - Adding or deleting a record
  - Changing the current record

Client scripts can be written from within the Web UI, there is a script editor built into the Business Components screen on the Form/Event Handlers panel.

More information on how to write these different event handler types can be found here: Client scripting

TypeScript for event handlers run only on the client side, while business documents only execute code directly on the server itself.

It's important to keep this in mind because the interaction on the Web UI could be different than the direct API interaction. For example, one can write OOABL code to limit acceptable data ranges and send an error message and undo, and this will work through a business document API, rendering the same error and behavior as on the Web UI itself. However, if one attempts to do the same with Typescript Event Handler, the limitation, error message, "and undo or reset to previous value" message only work on the Web UI, while sending the invalid value through the API does not trigger the Event Handler.

Server-side scripts

Server scripts are running on a JavaScript engine on the server. They are an extension to Business Objects. Server scripts can be written for platform business components as well as for standard QAD business components. Server scripts can override public Business Component functions and to serve actions such as:

- Start/commit/rollback transactions
- Modify the business component data
- Add extra validations
- Call other Business Components
- Translate a string
- Get current user/role information
- Query the database

Server scripts need to be developed outside the Web UI, with a TypeScript code editor.

How that is done is explained in detail here: Server scripting using TypeScript

Client scripting

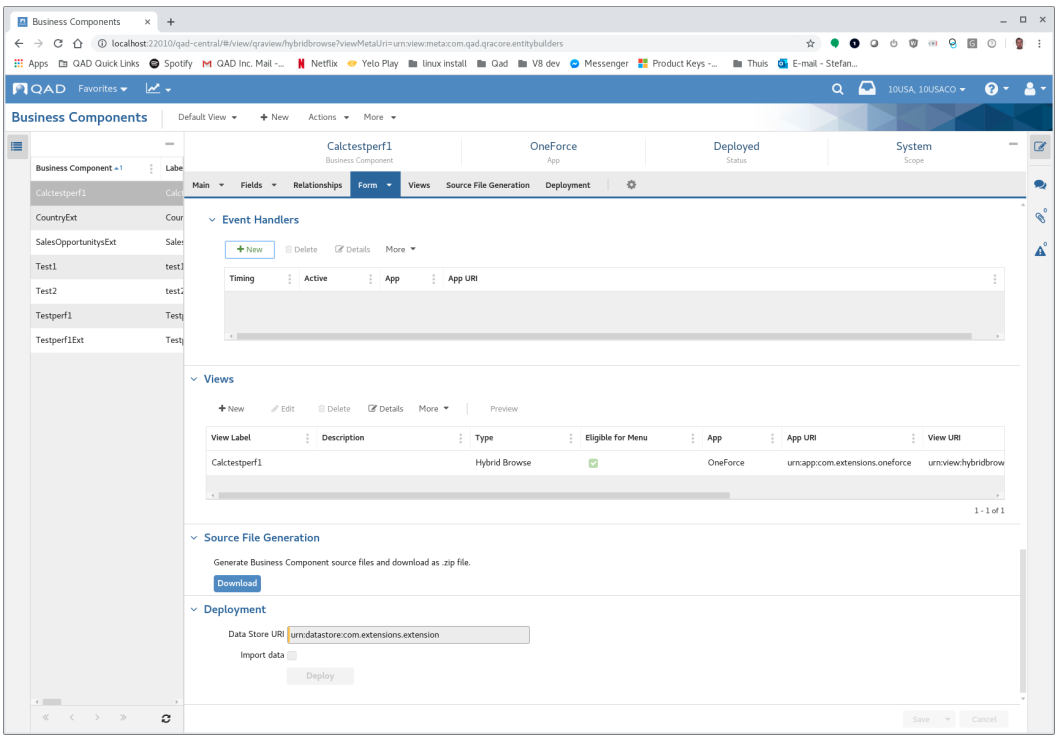
Introduction

Client scripts are running on the client, in the browser. They can only act on UI events of the maintenance view, not the browse part of a hybrid view. A client script can act on different parts of the maintenance screen by attaching a certain type of event handler to the screen. These different handler types can be created inside the script.

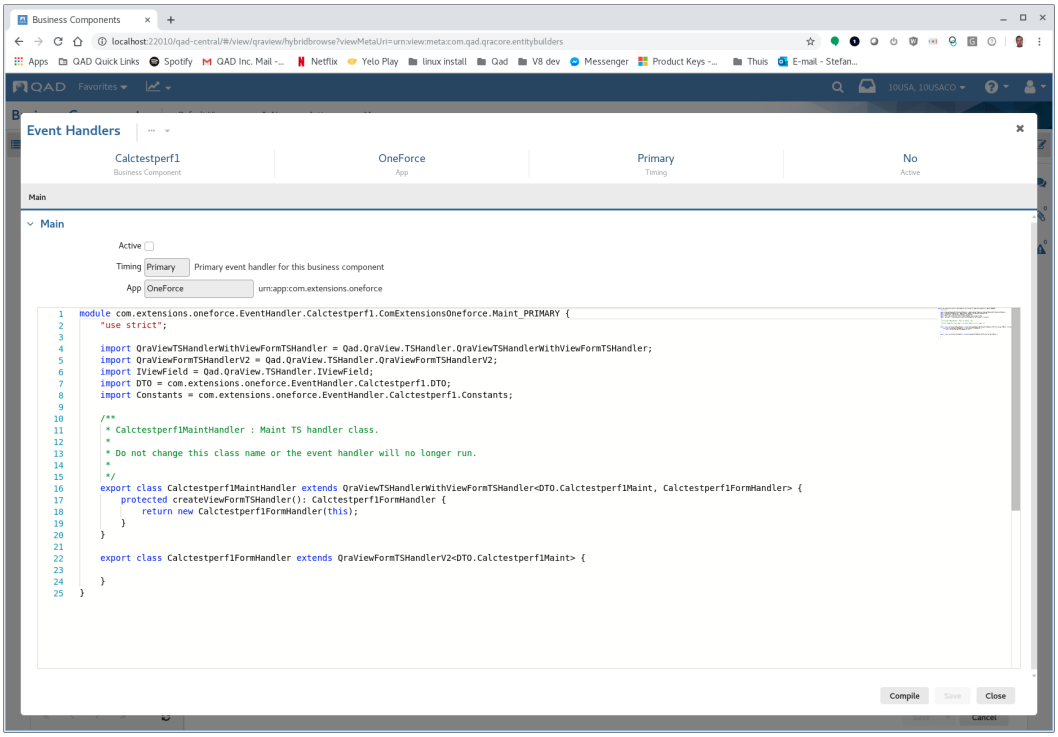
This page describes how these different handlers are written and what can be done with them.

Script editor

The client script editor can be found on the business components maintenance UI, under the forms/UI event handlers section:



You can click New to create a new event handler, and then the following screen with the editor appears:



The following flags can be set to the script:

- Active: when set, the script will be active at runtime. If not set, the script will not run. This is mainly used for debugging purposes.
- Timing: this determines when the script will run. It is different for Business Components that belong to the current App than for Business Components belonging to another app.
    - Business Components belonging to another app than the current developer app: there are 2 timing options:
        - Pre: this means that the logic in this UI event handler will run BEFORE the logic of the other app
        - Post: this means that the logic in this UI event handler will run AFTER the logic of the other app
    - Business Components belonging to the same app as the current developer app: there is no choice, it is always:
        - Primary: this is the primary script for the Business Components. Other apps will be able to run scripts before or after this logic.

Note that scripts are written for the current developer app. This app can be set in the My Developer Settings UI.

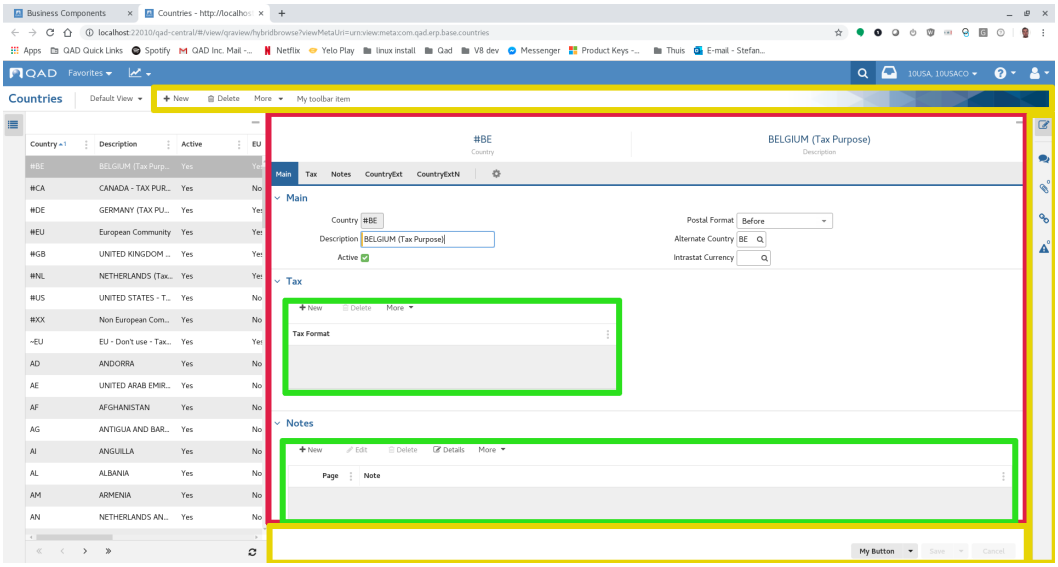
The content of the editor is an initial script that contains some empty classes to start with.

Initial script

The initial content of the editor is an initial script that contains the class declarations for the main UI event handler. Details can be found on the Main view UI event handler page. Good to know is that you can regenerate this code, if you clear out everything from the editor, and then save and close the editor pop-up window. Then, when reopening, the code will be regenerated.

Event handler types

Event handling in the UI is done with different objects handling different parts of the UI. These different objects are instances of TypeScript classes inside the UI event handler script. The following areas on the UI are handled by different handler objects. In the below diagram, you see the different areas:



So, a client script is actually a set of TypeScript classes that inherit from UI handler base classes. The base class that is inherited from determines the type of the UI event handler. There are three different types:

Main view UI event handler

This is the event handler that is instantiated automatically when the UI is opened. It can handle all the main UI events like when the UI is initialized, when the data is loaded, when it is saved, etc. This handler is also responsible for instantiating the other desired event handlers that can handle form and grid events.

This handler can handle data events (data loading, saving, etc.) and the areas in the yellow rectangles in the picture above.

Form UI event handler

This is an event handler that can handle form events such as field change, field leave, form button click, etc. There is only one (html)form on the UI. So, if form event handling is desired, there needs to be one form event handler class in the script.

This handler can handle form events (form fields, form buttons, etc.). This is the red areas in the picture above, minus the green areas.

Grid UI event handler

A grid event handler can handle grid events such as row change, cell modify, etc. If event handling for a grid on the UI is desired, then there needs to be a Grid event handler in the script. Typically, there is one grid handler class for every grid on the UI, but it can also be one class handling one or more grids on the screen.

This handler can handle grid events. The green areas in the picture above are grids for which events can be handled in this type of handler.

All three event handler types have access to other parts of the screen via the ViewController property on their base class. They also have access to the underlying data transfer objects that are used to fill the UI with data. More detail on this can be found here: UI event handler data transfer objects and UI element wrapper objects

Details on how to write these three event handler classes can be found here:

1. Main view UI event handler
2. Form UI event handler
3. Grid UI event handler

Inheritance model

UI event handler classes are all inheriting from a base class. That base class determines the type of the event handler. For example, a form event handler inherits from QraViewFormTSHandlerV2, a grid event handler from ViewGridTSHandlerV2, details are explained in the corresponding pages for the different UI event handler types. It is, however, important to know, that in these base classes there are methods with an empty implementation for every event that can be handled. If you look, for example, to the QraViewFormTSHandlerV2 base class code:

```
    export class QraViewFormTSHandler<TNgData> extends BaseQraViewTSHandler<TNgData> implements
IQraViewFormTSHandler<TNgData> {
        protected qraViewTSHandlerCallBack: IQraViewTSHandlerCallBack;
        /**
         * Constructor.
         */
        constructor(tsHandlerKey: string,qraViewController: IQraViewController<TNgData>,
qraViewTSHandlerCallBack: IQraViewTSHandlerCallBack,$scope: IQraViewControllerScope, $element:
JQuery,$timeout: ng.ITimeoutService,$modal: IModalService,focusController: IFocusController) {
            super(tsHandlerKey,qraViewController,$scope,$element,$timeout,$modal,focusController);
            this.qraViewTSHandlerCallBack=qraViewTSHandlerCallBack;
        }

        //field events
        /**
         * Called when the handler is added to the view field.
         * Initializing for the view field can be done here.
         */
        public onHandlerAddedToViewField(viewField: IViewField<any>): void {};
        /**
         * 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)".
         */
        public onFieldChange(viewField: IViewField<any>,eventData: Qad.Common.Service.
Communication.EventData.QraView.FieldChangeEventData<any>,processEvent: (processIt?: boolean) =>
void): void {};
        /**
         * Called when a field loses focus, regardless of any change to the field value.
         */
        public onFieldLeave(viewField: IViewField<any>,eventData: Qad.Common.Service.
Communication.EventData.QraView.FieldLeaveEventData<any>): void {};
        /**
         * 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.
         */
        public onFirstFieldChange(viewField: IViewField<any>,eventData: Qad.Common.Service.
Communication.EventData.QraView.FirstFieldChangeEventData): void {};
        /**
         * Called when a button defined in the view meta data is clicked.
         */
        public onButtonClick(viewButton: IViewButton, eventData: Qad.Common.Service.Communication.
EventData.QraView.ButtonClickEventData): void {};
        /**
         * Called when a field defined in the view meta data that supports ng-click is clicked.
         */
        public onFieldClick(viewField: IViewField<any>, eventData: Qad.Common.Service.
Communication.EventData.QraView.FieldClickEventData): void {};
        /**
         * Method to cleanup when the handler is destroyed.
         */
        protected onDestroy():void {
        }

        protected _onDestroy(): void {
            super._onDestroy();
            delete this.qraViewTSHandlerCallBack;
        }

    }

    export class QraViewFormTSHandlerV2<TNgData> extends QraViewFormTSHandler<TNgData> {
        constructor(creator: TSHandlerCreator<TNgData>) {
            var creatorData=creator.getTSHandlerCreatorData();
            super(creatorData.tsHandlerKey,creatorData.qraViewController,creatorData.
qraViewTSHandlerCallBack,creatorData.$scope,creatorData.$element,creatorData.$timeout,
creatorData.$modal,creatorData.focusController);
        }

    }
```

You can see that there are functions for all the possible UI events that can occur. These methods are being called at runtime when a UI event occurs, and what you do in your UI event handler class is overriding this method with your implementation. So that at runtime, your implementation is called instead of then empty implementation of the parent.

Overriding a function from a base class in TypeScript simply means that you have to declare the same function in your derived class:

```
    export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {
        onFieldChange(viewField: IViewField<any>, eventData: EventData.QraView.
FieldChangeEventData<any>, processEvent: (processIt?: boolean) => void): void {
            //your implementation for the field change event
        }
    }
```

Now, if you want to implement an event handler for a certain event function, you need to have exactly the same declaration of the function you are overriding. You can look up that method declaration in the documentation, but you can also use this little trick with the editor: UI event handler development tips and tricks, Overriding an event handler base class method

JavaScript frameworks and libraries

The Web UI is built with some 3rd party libraries. These are the most important ones:

- AngularJS
- KendoUI
- JQuery

These libraries are exposed to the UI event handlers and can be used directly. However, it is not recommended because when we upgrade the libraries or replace them with other libraries, some UI event handlers might be broken. Especially using AngularJS or KendoUI directly is dangerous. JQuery is less dangerous. It should, however, be avoided to directly manipulate the html with JQuery or to search for elements by their ID. Instead, use the QAD TypeScript api to get to the HTML elements.

Debugging client scripts

Client scripts can be debugged with the Chrome developer tools. How that is done is described here: Debugging UI event handlers

UI event handler data transfer objects and UI element wrapper objects

Introduction

This page describes the script objects that are available for a UI event handler developer to get access to the UI's underlying data and UI elements.

MVC model

The UI's and event handler runtime are implemented in a MVC way. So, we have a View (the html), a model (the DTO objects with the data), and a View Controller (containing the UI logic). This can be seen in the structure of the UI event handler base classes and the objects it references:

The model

The model can be accessed with the NgData property on the base class:

```
    export class Calctestperf1FormHandler extends QraViewFormTSHandlerV2<DTO.Calctestperf1Maint> {
        public onFieldLeave(viewField: IViewField<any>, eventData: EventData.QraView.
FieldLeaveEventData<any>): void {

            let idfield1=this.NgData.calctestperf1s[0].idfield1;
        }
    }
```

As you can see in the above example, the NgData property gives you access to the one record that is loaded in the UI. Only for a grid event handler, the NgData contains something different. It contains an array of records that are displayed in the grid:

```
    export class Relation1Grid extends Qad.QraView.TSHandler.ViewGridTSHandlerV2<DTO.CountryMaint,
DTO.Countries01_Relation1,DTO.Cntr01ext13080747 | kendo.data.ObservableObject> {
        onAutoGridAfterInit(eventData: EventData.AutoGrid.AfterInitEventData) {
            //access the first record in the grid :
            let addedCurrency=this.NgData.Countries01_Relation1[0].addedCurrency;
        }
    }
```

Data Transfer Object Types

As you can see in the above examples, we use TypeScript generics to set the type of the data access properties of UI objects to the correct DTO class or interface. For example, this line:

```
export class Calctestperf1FormHandler extends QraViewFormTSHandlerV2<DTO.Calctestperf1Maint> {
```

uses generics to set the type of the NgModel property to the DTO.Calctestperf1Maint class. This way we have type checking on DTO objects. The DTO classes are automatically generated and accessible in the DTO namespace. A DTO alias is automatically generated in the initial script:

```
import DTO = com.extensions.oneforce.EventHandler.Calctestperf1.DTO;
```

The view controller

The view controller is accessible with the ViewController property:

```
    export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {
        onFieldLeave(viewField: IViewField<any>, eventData: EventData.QraView.
FieldLeaveEventData<any>): void {
            this.ViewController.refreshDataGrids(true);
        }
    }
```

In the above example, you see that the data grids on the screen are refreshed by calling the refreshDataGrids method on the view controller.

The view

The view is actually the html displayed in the browser. However, it's a very bad idea to access the html directly (because we are using libraries that generate the html). Accessing UI elements should be done by using the wrapper objects we have available on the view controller. The different wrapper objects that are available are:

ViewField

A view field is a wrapper object around a form field. It's always a combination of a label and an input box (although the label can be hidden):

```
this.ViewController.getViewField("MyViewField").IsReadOnly=true;
```

ViewLabel

A view label is just a text on the UI. It's a separate label without a field:

```
this.ViewController.getViewLabel("MyViewLabel").IsVisible=false;
```

ViewGrid

A view grid is a wrapper around a data grid on the screen:

```
this.ViewController.getViewGrid("MyViewGrid").DetailsLinkEnabled=false;
```

GroupPanelNavigator and GroupPanel

The group panel navigator is the object that wraps the panel navigator on top of the screen. The group panels on the screen are wrapped by a GroupPanel object, and accessible through the GropePanelNavigatorObject:

```
let hasChildren=this.ViewController.GroupPanelNavigator.getGroupPanelByName("MyGroupPanel",true).
HasVisibleChildGroupPanel;
```

ViewButton

A ViewButton is a form button (not a button from the bottom bar). It can be accessed as follows:

```
this.ViewController.getViewButton("MyButton").IsDisabled=true;
```

ErrorGroupPanel

The error group panel is the panel that appears at the bottom of the screen when a business logic error is returned from the server. This panel can be manipulated in the event handler:

```
this.ViewController.ErrorGroupPanel.clearErrorGrid();
```

There are other parts of the UI that can be accessed in the event handler. However, they don't have wrapper objects. Instead, there are global functions or view controller functions that can be used to manipulate them. What is available for the different screen elements is listed here: UI elements list of events and Properties/Functions

Main view UI event handler

Introduction

This UI event handler is the main event handler that is needed if you want to handle UI events. It is automatically instantiated when the UI is opened, and it is responsible for creating the other desired event handlers such as the form and grid event handlers. This event handler is also the one that can handle all main UI events, such as button bar events, data events, initializing events, etc. This page describes how to write this main event handler.

Initial script

The first time you open the script editor, an initial script is generated for you:

```
module com.qad.erp.base.EventHandler.Country.ComExtensionsOneforce.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.Country.DTO;
    import Constants = com.qad.erp.base.EventHandler.Country.Constants;

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

    export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {

    }
}
```

As you can see, it contains a class for a main UI event handler and a form UI event handler class. If you want this code to be regenerated, then you have to clear the editor, save, close and reopen the editor pop-up window.

Class definitions

The initial script contains two class definitions, for the following event handlers:

Main view event handler

The class definition of the main UI event handler is as follows: export class CountryMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.CountryMaint, CountryFormHandler>

It inherits from QraViewTSHandlerWithViewFormTSHandler. This is a base class that has all the methods for handling main UI events, and it simplifies the creation of the form TS handler (see next paragraph, initializing).

As you can see in the class declarations, generics are used to add types to generic parts of the base class. For this class, the two types are:

1. DTO.CountryMaint: this is the type of the DTO object that holds the data presented on the UI. It is accessible by the property NgData on the base class (see UI event handler data transfer objects and UI element wrapper objects).
2. CountryFormHandler: this is the type of the form handler that is used in this main event handler. It can be accessed by the ViewFormTSHandlerproperty on the base class.

Form event handler

Details can be found here: Form UI event handler

Initializing and destroying

Initializing

When the UI is first loaded, this event handler is instantiated, and its init method is called:

```
    export class CountryMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
CountryMaint, CountryFormHandler> {
        protected createViewFormTSHandler(): CountryFormHandler {
            return new CountryFormHandler(this);
        }

                // the following is not automatically generated, but put here for clarity. It is
only necessary if you want to handle grid events.
        protected onViewGridCreated(viewGrid: Qad.QraView.TSHandler.IViewGrid<any, any, kendo.
data.ObservableObject>): void {
        }

        init() {
            //This method is called right after the constructor
        }
    }
```

Note that the init method is not generated. If you want to use it, you'll need to add it to the code.

After that, while the UI is loading, the following methods will be called:

1. createViewFormTSHandler: this method is called when the form is ready, and the form event handler can be attached.
2. createViewGridTSHandler: this method is called for every grid that is initialized, and in the list of grids that have an event handler. Note that this method is not automatically generated for you. You only need it if you want to handle grid events. You can find all the detail on the Grid UI event handler page).

After these calls, the event handlers in your code will be called when an event is occurring for which a method is implemented in your event handler.

Destroying

When the UI is being closed, an onDestroy method is called on every event handler:

```
    export class CountryMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
CountryMaint, CountryFormHandler> {
        protected createViewFormTSHandler(): CountryFormHandler {
            return new CountryFormHandler(this);
        }

        protected onDestroy() {
                //this method is called when the UI is destroyed.
        }
    }
```

This onDestroy event can be used to clean up whatever needs to be cleaned up (e.g. if you added something to the window object).

Note that the onDestroy method is not generated. If you want to use it, you'll need to add it to the code.

Instantiating Form and Grid event handlers

In the previous section, you saw that during initialization createViewFormTSHandler and createViewGridTSHandler is being called. These methods are being called to get the instance of the form event handler, and the grid event handlers that need to be attached. So, in these methods, you need to return an instance of the handler you want to handle the form and grid events. The createViewFormTSHandler code is already generated for you:

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

                // the following is not automatically generated, but put here for clarity. It is
only necessary if you want to handle grid events.
                protected onViewGridCreated(viewGrid: Qad.QraView.TSHandler.IViewGrid<any, any,
kendo.data.ObservableObject>): void {
        }

    }
```

As you can see, createViewFormTSHandler returns an instance of the form TS handler in your code (this was already generated).

The createViewGridTSHandler is not generated for you, you only need it when you need to handle grid events. How you implement that is described here: Grid UI event handler

Event handler functions

If you want your event handler to act on UI events, you'll need to override the event method from the base class that you want to implement. Here, you can see how you can easily find the methods you can override: Overriding an event handler base class method

The methods you can override are listed on the pages of the corresponding UI element that you can handle events for. The main event handler can handle events for the following UI elements:

1. Main View
2. Hybrid View
3. Error viewer (ErrorViewer)
4. Group Panel (GroupPanelNavigator and GroupPanel)
5. Toolbar

Form UI event handler

Introduction

This UI event handler is the event handler that is needed if you want to handle Form UI events. It is instantiated by the main UI event handler when the UI is opened. The form is everything between the top toolbar and the bottom toolbar. So, this event handler can handle form events such as form field events (field change), form button events (button click), etc. This page describes how to write this event handler.

Initial script

The first time you open the script editor, an initial script is generated for you, the form event handler is part of that:

```
module com.qad.erp.base.EventHandler.Country.ComExtensionsOneforce.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.Country.DTO;
    import Constants = com.qad.erp.base.EventHandler.Country.Constants;

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

    export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {

    }
}
```

As you can see it contains a class for a main UI event handler and a **form UI event handler class**. If you want this code to be regenerated, then you have to clear the editor, save, close and reopen the editor pop-up window.

Class definitions

The initial script contains two class definitions, for the following event handlers:

Main view event handler

This is described here: Main view UI event handler

Form event handler

For your convenience, also the form event handler is already generated: export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint>

It inherits from QraViewFormTSHandlerV2, that's a base class that has all the methods for handling form events.

Also for this class generics are used. There is one generics parameter:

1. DTO.CountryMaint: this is the type of the DTO object that holds the data presented on the UI. It is accessible by the property NgData on the base class (see UI event handler data transfer objects and UI element wrapper objects).

If your form also contains grid, and if you want to handle grid events, you'll have to add grid handler classes too. More detail for this can be found here: Grid UI event handler

Initializing and destroying

Initializing

When the UI is first loaded, this event handler is instantiated from the main event handler. So, initialization code should go in the constructor:

```
export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {
    constructor(creator: Qad.QraView.TSHandler.TSHandlerCreator<DTO.CountryMaint>) {
        super(creator);
                    //your init code here
    }
}
```

Note that the constructor is not generated. If you want to use it, you'll need to add it to the code.

After this call, the event handlers in your code will be called when an event is occurring for which a method is implemented in your event handler.

Destroying

When the UI is being closed, an onDestroy method is called on every event handler, also on the form event handler:

```
export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {
    protected onDestroy() {

    }
}
```

This onDestroy event can be used to clean up whatever needs to be cleaned up (e.g. if you added something to the window object).

Note that the onDestroy method is not generated. If you want to use it, you'll need to add it to the code.

Event handler functions

If you want your event handler to act on UI form events, you'll need to override the event method from the base class that you want to implement. Here, you can see how you can easily find the methods you can override: Overriding an event handler base class method

The methods you can override are listed on the pages of the corresponding UI element that you can handle events for. The form event handler can handle events for the following UI elements:

1. Form Field (ViewField)
2. Form Button (ViewButton)

Grid UI event handler

Introduction

This UI event handler is the event handler that is needed if you want to handle Grid UI events. It is instantiated by the main UI event handler when the UI is opened. For every grid on the screen, you can add a grid event handler. So, this event handler can handle grid events such as add row event, change active row event, etc. This page describes how to write this event handler.

Initial script

The first time you open the script editor, an initial script is generated for you. Grid handlers are not part of this script, you'll need to add them for every grid you want to handle events for:

```
module com.qad.erp.base.EventHandler.Country.ComExtensionsOneforce.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.Country.DTO;
    import Constants = com.qad.erp.base.EventHandler.Country.Constants;

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

    export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {

    }
}
```

As you can see it contains a class for a main UI event handler and a **form UI event handler class**. If you want this code to be regenerated, then you have to clear the editor, save, close and reopen the editor pop-up window.

Class definition

Grid event handlers class definitions need to be added manually like in these examples:

export class CountryVatFormatAutoGridHandler extends Qad.QraView.TSHandler.ViewGridTSHandlerV2<DTO.CountryMaint,DTO.CountryVatFormat[],DTO.CountryVatFormat>

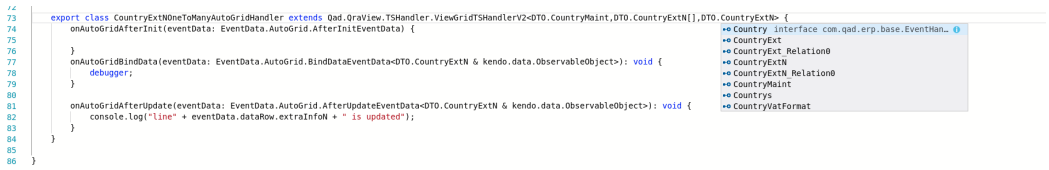
export class CountryExtNOneToManyAutoGridHandler extends Qad.QraView.TSHandler.ViewGridTSHandlerV2<DTO.CountryMaint,DTO.CountryExtN[],DTO.CountryExtN>

As you can see, the class declaration also uses generics for the DTO class types. In this case, there are two DTO types. If we take the above examples, it is as follows:

1. CountryVatFormatAutoGridHandler:
    a. DTO.CountryMaint: this is the type of the general data object the UI is working with. It is the same as for the main UI event handler and the form UI event handler, it determines the type of this.NgData.
    b. DTO.CountryVatFormat[]: this is the type of the data object the grid is bound to. It is an array of record types. In this case, an array of Country Vat Format records.
    c. DTO.CountryVatFormat: this is the type of one record in the grid. In this case, it is a country vat format record.
2. CountryExtNOneToManyAutoGridHandler:
    a. DTO.CountryMaint: this is the type of the general data object the UI is working with. It is the same as for the main UI event handler and the form UI event handler, it determines the type of this.NgData.
    b. DTO.CountryExtN[]: this is the type of the data object the grid is bound to. It is an array of record types. In this case, an array of Country ExtN records. ExtN is an extension to Country (Many to one).
    c. DTO.CountryExtN: this is the type of one record in the grid. In this case, it is a record from the Country ExtN extension.

As you can see in the examples, the first type is the type of the maint DTO class, you can **copy** it from this line: export class CountryMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<**DTO.CountryMaint**, CountryFormHandler>

The second parameter is the data object the grid is bound to. It is always an array of the 3rd types. The 3rd type is the DTO type of one record. The way to find that DTO type, is through the name. If you type "DTO.", it will show you a list of classes:



From this drop down, you have to pick the correct one. You can do that easier by looking into the DTO source code. See UI event handler editor tips and tricks, Finding the grid DTO type.

Initializing and destroying

Initializing

When the UI is first loaded, this event handler is instantiated from the main event handler. So, initialization code should go in the constructor:

```
export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {
    constructor(creator: Qad.QraView.TSHandler.TSHandlerCreator<DTO.CountryMaint>) {
        super(creator);
                    //your init code here
    }
}
```

Note that the constructor is not generated. If you want to use it, you'll need to add it to the code.

After this call, the event handlers in your code will be called when an event is occurring for which a method is implemented in your event handler.

Destroying

When the UI is being closed, an onDestroy method is called on every event handler, also on the form event handler:

```
export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {
    protected onDestroy() {

    }
}
```

This onDestroy event can be used to clean up whatever needs to be cleaned up (e.g. if you added something to the window object).

Note that the onDestroy method is not generated. If you want to use it, you'll need to add it to the code.

Instantiating Grid event handlers

On the page Main view UI event handler, you saw that during initialization createViewFormTSHandler and createViewGridTSHandler are being called. These methods are being called to get the instance of the form event handler, and the grid event handlers that need to be attached. So, for all the grids you want to handle events for, you need to return an instance of the handler class you created for it, like in this example:

```
module com.qad.erp.base.EventHandler.Country.ComExtensionsOneforce.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.Country.DTO;
    import Constants = com.qad.erp.base.EventHandler.Country.Constants;

    /**
     * CountryMaintHandler : Maint TS handler class.
     *
     * Do not change this class name or the event handler will no longer run.
     *
     */
    export class CountryMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
CountryMaint, CountryFormHandler> {
        private countryVatFormatAutoGrid:CountryVatFormatAutoGridHandler;
        private countryExtNOneToManyAutoGrid:CountryExtNOneToManyAutoGridHandler;

        protected init() {
            this.ViewGridsToHandleList=["CountryVatFormatAutoGrid","
countryExtNOneToManyAutoGrid"];
        }
        protected createViewFormTSHandler(): CountryFormHandler {
            return new CountryFormHandler(this);
        }

        protected createViewGridTSHandler(viewGrid: Qad.QraView.TSHandler.IViewGrid<any, any,
kendo.data.ObservableObject>): Qad.QraView.TSHandler.ViewGridTSHandler<any, any, any, any> {
            if (viewGrid.GridID==="CountryVatFormatAutoGrid") {
                this.countryVatFormatAutoGrid=new CountryVatFormatAutoGridHandler(viewGrid,this);
                return this.countryVatFormatAutoGrid;
            } else if (viewGrid.GridID=="countryExtNOneToManyAutoGrid") {
                this.countryExtNOneToManyAutoGrid=new CountryExtNOneToManyAutoGridHandler
(viewGrid,this);
                return this.countryExtNOneToManyAutoGrid;
            }
        }

    }

    export class CountryFormHandler extends QraViewFormTSHandlerV2<DTO.CountryMaint> {
    }

    export class CountryVatFormatAutoGridHandler extends Qad.QraView.TSHandler.
ViewGridTSHandlerV2<DTO.CountryMaint,DTO.CountryVatFormat[],DTO.CountryVatFormat> {
    }

    export class CountryExtNOneToManyAutoGridHandler extends Qad.QraView.TSHandler.
ViewGridTSHandlerV2<DTO.CountryMaint,DTO.CountryExtN[],DTO.CountryExtN> {
    }

}
```

As you can see, createViewGridTSHandler returns instances for the grid event handlers. The method createViewGridTSHandler will ONLY be called for the grids that are in the array ViewGridsToHandleList property of the base class.

Note that the grid event handlers are also assigned to a class variable. This is not mandatory, it is done in case you want to call one of these grid handlers elsewhere in the main handler code.

Event handler functions

If you want your event handler to act on UI form events, you'll need to override the event method from the base class that you want to implement. Here, you can see how you can easily find the methods you can override: Overriding an event handler base class method

The methods you can override are listed on the pages of the corresponding UI element that you can handle events for. The form event handler can handle events for the following UI elements:

1. Data Grid (ViewGrid)

UI elements list of events and Properties /Functions

Introduction

This section describes all the possible events, properties, and functions a developer has available to manipulate the behavior of certain parts of the screen. It is divided per UI control. Every child page describes all the possibilities you have for that control.

UI elements:

Form Field (ViewField)

Getting access to a ViewField object

A form field is accessible through an object of type IViewField. Getting access to such an object (called view field) can be done in the following ways:

1) Through the view controller. In every event handler class, you can do the following:

```
let taxClassViewField=this.ViewController.getViewField("TaxClassAutoField");
```

2) Using the object reference passed in an event. The following code shows that the view field object is passed in the onFieldChange event:

```
    export class SbjTest1FormHandler extends QraViewFormTSHandlerV2<DTO.SbjTest1Maint> {
        public onFieldChange(viewField: IViewField<any>, eventData: EventData.QraView.
FieldChangeEventData<any>, processEvent: (processIt?: boolean) => void): void {
            if (viewField.FieldName==="IsTaxableAutoField") {
                if (viewField.Value==="abc") {
                    viewField.IsReadOnly=true;
                }
            }
        }
    }
```

Events

Form fields events are handled in a Form UI event handler. The following functions can be implemented to handle form field events:

Method	Event Data Properties	Description
onButtonClick (viewButton: IViewButton,eventData: Button.ButtonClickEventData)	null	Called when a button defined in the view metadata is clicked or when a button defined using addBotto mButton() is clicked.
onFirstFieldChange		Called when the first field in the form is changed in a standalone (non-data-grid) field. Fired as soon

(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	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.

Properties/Functions

Show Hide a Field/Add Remove Field from config panel

Showing or hiding a field can be done in three ways:

1) Setting the property IsVisible on the view field object:

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

2) Using the setControlVisible method:

```
setControlVisible("TaxClassAutoField",this.$element,false,true,true);
```

This method should be used if you want to control how the UI behaves when the field is set invisible:

setControlVisible(Id of the field, Starting Element, Want it visible, Want to adjust field layout);

Parameters:

- startElement: HTML element under which to find the field
- isVisible: boolean indicating whether field should be displayed or hidden, * true --> show & false --> hide
- adjustLayout: boolean to adjust layout of fields, will fill gaps or move all fields to left if * all fields in left side are hidden. Default value is True.
- updateDisabled: boolean to disable checkbox for specified field in config panel, so a user can't override it. Default value is True.

3) Using show/hide functions:

```
this.ViewController.getViewField("TaxClassAutoField").hide(true);
```

Parameters:

- adjustLayout: boolean to adjust layout of fields, will fill gaps or move all fields to left if * all fields in left side are hidden. Default value is False.

```
this.ViewController.getViewField("TaxClassAutoField").show();
```

Set the value of a Field

Setting the value can be done with the Value property.

```
this.ViewController.getViewField("TaxClassAutoField").Value="abc";
```

Disable/Enable a Field

Disabling or enabling a Field can be done with the IsDisabled property:

```
this.ViewController.getViewField("TaxClassAutoField").IsDisabled=true;
```

Disable/Enable a Multiple Fields of Group Panel/Html Container

Use ViewController.setControlsDisabled to enable/disable multiple child elements.

Here is an example that shows how to enable/disable all elements inside the group panel. This method disables all input elements, sub-panels, and data grids.

```
this.ViewController.setControlsDisabled($("#InventoryDataGroupPanel",this.$element), true);
```

Set field as Readonly/Enabled

Making a Field read-only can be done with the IsReadOnly property:

```
this.ViewController.getViewField("TaxClassAutoField").IsReadOnly=true;
```

ReadOnly is different from disabled, you can still put a cursor in the field to copy the value.

Set field as Required

Making a Field required (yellow bar in front of a field) can be done with the IsRequired property:

```
this.ViewController.getViewField("TaxClassAutoField").IsRequired=true;
```

Set Placeholder

Use the PlaceholderText property to get/set placeholder for a field:

```
this.ViewController.getViewField("TaxClassAutoField").PlaceholderText="some placeholder";
```

Not applicable for dropdowns and checkboxes.

Get json path

Use the Binding property to get field json path:

```
this.ViewController.getViewField("TaxClassAutoField").Binding;
```

Show/Hide lookup button

Use the ShowLookupButton property to show or hide the lookup button of a field with lookup:

```
this.ViewController.getViewField("TaxClassAutoField").ShowLookupButton=true;
```

Show/Hide Drop-down-list Items

Use ViewField.setListItemsVisible to hide/show specific list control items:

setListItemsVisible(listItems: any, isVisible: boolean, value?: any)

Parameters:

- listItems: single value or list of values to show or hide * e.g. "val1" or ["val1", "val2"]
- isVisible: boolean to show list item(true) or hide list item (false)
- value (optional): current selected value of the drop-down list.

Example

```
this.ViewController.getViewField("TaxClassAutoField").setListItemsVisible(["v3","v4"], true,
"v4");        //show list items of values "v1" & "v2"        and set current selected value to
"v4"
```

Dynamically Replace Drop-down-list List Items

Use ViewField.setListItems to replace the drop-down-list list items:

setListItems(listItems: any, valueToSelect?: any)

Parameters:

- listItems: array of list items for the list control in the format: [{dataLabel: "Some Label", dataValue: "foo"}, {dataLabel: "Another Label, dataValue: "bar"}] (must contain at least one item)
- valueToSelect: optional parameter, defaults to null – the value that should be selected in the list after the control's list items are reset, if null, the first item will be selected.

Here is an example, Lot Serial Control drop-down list in which the UI event handler is changing the List Items dynamically.

```
var listItems = [{dataLabel: "Some Label", dataValue: "foo"}, {dataLabel: "Another Label,
dataValue: "bar"}];
var valueToSelect = "foo";
this.ViewController.getViewField("LotSerialControlAutoField").setListItems(listItems,
valueToSelect);
```

Check if Drop-down-list List is visible

Checking if the drop-down-list List is currently visible can be done in two ways:

1) Using a 'visible' selector:

```
this.ViewController.getViewField("TaxClassAutoField").Element.data("kendoDropDownList").wrapper.is
(":visible")
```

2) Using the IsDropDownVisible property:

```
this.ViewController.getViewField("TaxClassAutoField").IsDropDownVisible
```

Form Button (ViewButton)

Getting access to a ViewButton object

A form field is accessible through an object of type IViewButton. Getting access to such an object (called view button) can be done in the following ways:

1) Through the view controller. In every event handler class, you can do the following:

```
let myButton=this.ViewController.getViewButton("MyButton");
```

2) Using the object reference passed in an event. The following code shows that the view field object is passed in the onButtonClick event:

```
onButtonClick(viewButton: Qad.QraView.TSHandler.IViewButton, eventData: EventData.QraView.
ButtonClickEventData): void {
    if (viewButton.Name==="MyButton") {
        //do something
    }
}
```

Events

Form button events are handled in a Form UI event handler. The following functions can be implemented to handle form field events:

Method	Event Data Properties	Description
onButtonClick (viewButton: IViewButton, eventData: Button. ButtonClickEventData)	null	Called when a button defined in the view metadata is clicked or when a button defined using addBottomButton() is clicked.

Properties/Functions

Show Hide a Button

Showing or hiding a button can be done setting the property IsVisible on the view button object:

```
this.ViewController.getViewButton("MyButton").IsVisible=false;
```

Disable/Enable a Button

Disabling or enabling a button can be done with the IsDisabled property:

```
this.ViewController.getViewButton("MyButton").IsDisabled=true;
```

Data Grid (ViewGrid)

Getting access to a ViewGrid object

A form field is accessible through an object of type IViewGrid. Getting access to such an object (called view grid) can be done in the following ways:

1) Through the view controller. In every event handler class, you can do the following:

```
let myGrid=this.ViewController.getViewGrid("MyGrid");
```

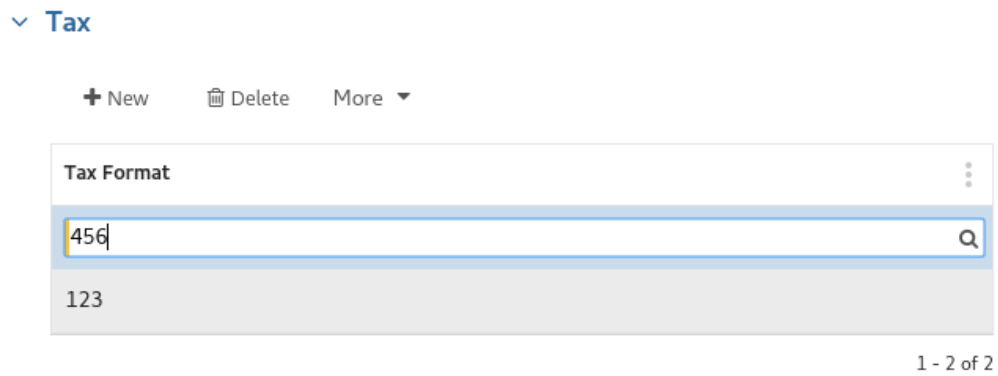
2) Using this.ViewGrid property of the grid UI event handler:

```
export class Relation1Grid extends Qad.QraView.TSHandler.ViewGridTSHandlerV2<DTO.CountryMaint,DTO.
Countries01_Relation1,DTO.Cntr01ext13080747 | kendo.data.ObservableObject> {
    onAutoGridAfterInit(eventData: EventData.AutoGrid.AfterInitEventData) {
        this.ViewGrid.DeleteButtonEnabled=false;
    }
}
```

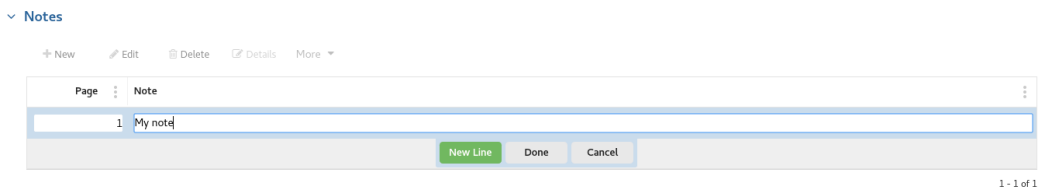
Difference between multi-line edit grid (internal grid) and single-line edit grid (external grid)

In the framework, we have two types of grid. There are multi-line edit grids and single-line edit grids.

The first one looks like this while editing:



The second one like this:



As you can see, the second one's line needs to be closed before going to the next line. The reason is that every line edit is running in its own transaction. The data in such a grid is coming from a separate Business Component, the grid is retrieving that data from that BC separately from the main UI itself (separate REST calls). So, after editing a line, the data for that line is sent to the corresponding BE with a rest call.

The first gird, the multi-line edit grid is edited within the same transaction as the rest of main UI. It displays a table that belongs to the same BC or a table of an embedded BC. So, this grid is not getting its data from separate rest calls, the data is coming along and saved with the data of the main BC of the screen.

You will see in the list of events that some of the events only apply to one of the two types of grids.

Grid data source (new)

Adding a record to the grid

```
    SomeGridHandler {
                private addRecords() {
                        this.ViewGrid.DataSource.addRecord({fild1: 'value1', field2: 'value2'
...});
                        this.ViewGrid.DataSource.addRecord({fild1: 'value3', field2: 'value4'
...});

                        // To sync data with NgData for the internal grid or to sync data with
server for the external grid
                        this.ViewGrid.DataSource.sync().then(() => {/*Do something after*/}); //
OR: await this.ViewGrid.DataSource.sync() OR if you do not do something after: this.ViewGrid.
DataSource.sync();
                }
         }
```

Adding a record to the internal sub-grid (It does not work for the external sub-grid)

```
const parentRow1 = {childPath: 'someChildPath', record: {keyField1: 'value', keyField2:
'value'}};
const childRowOfParentRow1 = {childPath: 'someChildPath2', record: {keyField1: 'value3',
keyField2: 'value3'}};

this.ViewGrid.DataSource.addRecord({fild1: 'value1', field2: 'value2' ...}, [parentRow1,
childRowOfParentRow1, ...]);
this.ViewGrid.DataSource.addRecord({fild1: 'value3', field2: 'value4' ...}, [parentRow1,
childRowOfParentRow1, ...]);

// To sync data with NgData for the internal grid
this.ViewGrid.DataSource.sync().then(() => {/*Do something after*/}); // OR: await this.ViewGrid.
DataSource.sync() OR if you do not do something after: this.ViewGrid.DataSource.sync();
```

The second parameter of **this.ViewGrid.DataSource.addRecord** is an array that contains a way to this sub-grid. That is a way from the current grid to its sub-child in this array. An item of this array contains properties: record and childPath.

- **record** is a search criteria object. Properties are just fields of the row which we want to find. It is enough and recommended to use only key-fields which help to identify the row that we need but we can use more fields and **it will execute longer.**
- **childPath** is a path to the sub-grid's data.

Inserting a record into the internal grid (It does not work for the external grid)

```
const position1 = 0;
const position2 = 5;

this.ViewGrid.DataSource.insertRecord(position1, {field1: 'ew12', field2: 'sas'})
this.ViewGrid.DataSource.insertRecord(position2, {field1: 'ew', field2: 'ddd'})

// To sync data with NgData for the internal grid
this.ViewGrid.DataSource.sync().then(() => {/*Do something after*/}); // OR: await this.ViewGrid.
DataSource.sync() OR if you do not do something after: this.ViewGrid.DataSource.sync();
```

Inserting a record into the internal sub-grid (It does not work for the external sub-grid)

```
const parentRow1 = {childPath: 'someChildPath', record: {keyField1: 'value', keyField2:
'value'}};
const childRowOfParentRow1 = {childPath: 'someChildPath2', record: {keyField1: 'value3',
```

```
keyField2: 'value3'}};
const position1 = 0;
const position2 = 5;

this.ViewGrid.DataSource.insertRecord(position1, {fild1: 'value1', field2: 'value2' ...},
[parentRow1, childRowOfParentRow1, ...]);
this.ViewGrid.DataSource.insertRecord(position2, {fild1: 'value3', field2: 'value4' ...},
[parentRow1, childRowOfParentRow1, ...]);

// To sync data with NgData for the internal grid
this.ViewGrid.DataSource.sync().then(() => {/*Do something after*/}); // OR: await this.ViewGrid.
DataSource.sync() OR if you do not do something after: this.ViewGrid.DataSource.sync();
```

The third parameter of **this.ViewGrid.DataSource.insertRecord** is the same as **the second parameter of this.ViewGrid.DataSource.addRecord**

Deleting a record from the grid

```
this.ViewGrid.DataSource.removeRecord({keyFiedl1: 'someValue1', keyField2: 'someValue2'});
//Deleting from the internal subgrid (It does not work for the external subgrid)
this.ViewGrid.DataSource.removeRecord({keyFiedl1: 'someValue3', keyField2: 'someValue4'}, []);

// To sync data with NgData for the internal grid or to sync data with server for the external
grid
this.ViewGrid.DataSource.sync().then(() => {/*Do something after*/}); // OR: await this.ViewGrid.
DataSource.sync() OR if you do not do something after: this.ViewGrid.DataSource.sync();
```

The first parameter of the method is a search criteria object. Properties are just fields of the row which we want to find. It is enough and recommended to use only key-fields which help to identify the row that we need but we can use more fields and **it will execute longer.**

The second parameter of **this.ViewGrid.DataSource.removeRecord** is the same as **the second parameter of this.ViewGrid.DataSource.addRecord**

Getting records from the internal grid (It does not work for the external grid)

```
const records1 = this.ViewGrid.DataSource.getRecords();
// Subgrid
const parentRow1 = {childPath: 'someChildPath', record: {keyField1: 'value', keyField2: 'value'}};
const childRowOfParentRow1 = {childPath: 'someChildPath2', record: {keyField1: 'value3',
keyField2: 'value3'}};
const records2 = this.ViewGrid.DataSource.getRecords([parentRow1, childRowOfParentRow1 ]);
```

The first parameter of **this.ViewGrid.DataSource.getRecords** is the same as **the second parameter of this.ViewGrid.DataSource.addRecord**

Refreshing record of the external sub-grid (It does not work for the internal sub-grid)

```
this.ViewGrid.DataSource.refresh().then(() => {/*Do something after*/});
// OR
await this.ViewGrid.DataSource.refresh();
/*Do something after*/
//OR if you do not do something after
this.ViewGrid.DataSource.refresh();
```

Grid data source (old)

A grid works with its own DTO objects. A multi-line edit grid works with a copy of a table in NgData. A single-line edit grid works with data it retrieved from the server with its own REST calls. For both grids, you can modify the data in your event handler. You need to modify that data in the dto objects of the grid, and after doing that, you can commit the data to the NgData object to the server.

Adding a record to the grid

```
        private addRecordToCountryVatFormatGrid() : void {
            let countryVatFormatGrid=this.ViewController.getViewGrid(Constants.DataGridNames.
CountryVatFormatAutoGrid);
            let countryVatFormatRecord:DTO.CountryVatFormat={countryCode:this.NgData.countrys[0].
countryCode,taxFormat:"MyFormat",
                concurrencyHash:undefined,dataOperation:undefined,lastModifiedDate:undefined,
lastModifiedTime:undefined,lastModifiedUser:undefined};

            countryVatFormatGrid.KendoGrid.dataSource.add(countryVatFormatRecord);
        }
```

Deleting a record from the grid

```
            let countryVatFormatGrid=this.ViewController.getViewGrid(Constants.DataGridNames.
CountryVatFormatAutoGrid);
            //delete currently selected record
            countryVatFormatGrid.KendoGrid.dataSource.remove(countryVatFormatGrid.
getSelectedRowData())
            countryVatFormatGrid.commitDataChanges();
```

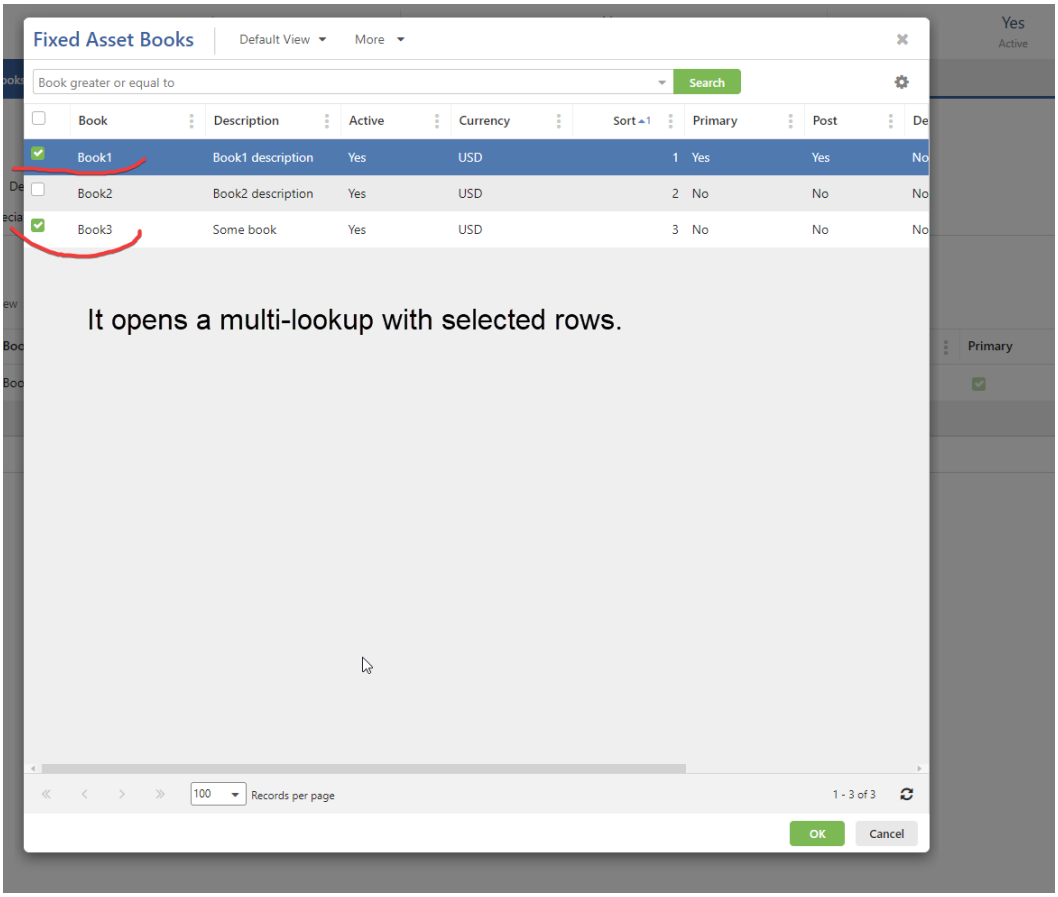
Committing the grid data

```
        private commitCountryVatFormatGrid() : void {
            let countryVatFormatGrid=this.ViewController.getViewGrid(Constants.DataGridNames.
CountryVatFormatAutoGrid);
            countryVatFormatGrid.commitDataChanges();
        }
```

Note that this is only necessary if you want to commit the changes immediately. The UI framework is doing that for you when you press the save button.

Launch Lookup Multi-Select (launchLookupMultiSelect)

```
              private showLookup() : void {
            const firstRowSerchConditionToSelect = {keyfield1: 'value1', keyfield2: 'value2'};
                const secondRowSerchConditionToSelect = {keyfield1: 'value3', keyfield2:
'value4'};
                const preSelectedRows = [firstRowSerchConditionToSelect,
secondRowSerchConditionToSelect]// as an example  [{'FABook.FABookCode': 'Book1'}, {'FABook.
FABookCode': 'Book3'}];
                const options = {lookupId: 'mfg:fa222', lookupSourceField: $('button'),
lookupSearchConditions:[{fieldName: 'field1', operatorEnum: 'ne', fieldValue: '',
fieldValueTypeEnum: ''}]};
                const isFilterVisible = true;

                this.ViewGrid.launchLookupMultiSelect(options, selectedRows, isFilterVisible);
        }

                public onAutoGridLookupComplete(event) {
                    console.log(event);    // All selected rows will be shown in the console
after selection in the popup
                  }
```



Events

Form button events are handled in a Grid UI event handler. The following functions can be implemented to handle 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 screen binds new data is bound. TODO: Commodity Code (in testp Update maint screen. The first tim just the correct details.
onAutoGridRowChange(eventData: EventData.AutoGrid.RowChangeEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row	Called whenever the selected row TODO: Need to support multiple r
onAutoGridFieldChangeEvent(eventData: EventData.AutoGrid.FieldChangeEventData<TGridRecordObservableRecord> , processEvent: (processIt?: boolean) => void)	**eventData.fieldName**: the name of the changed field	Called when an auto grid field valu focus is moved away from the cha Note: only fired in response to fiel data), not when fields are bound t

	eventData.fieldValue: 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	Note: in order to cancel the fieldCl the field), it is necessary for the su TODO: There is a tough-to-reprod thrown when trying to publish this. TypeError: Cannot read property 'i
onAutoGridFieldLeaveEvent(eventData: EventData.AutoGrid.FieldLeaveEventData)	**eventData.fieldName**: the ID of the field **eventData.fieldValue**: the current value of the field	Called when an auto grid field lose 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 button is clic Edit mode. TODO: Need to support multiple r

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 patch 1)
		
		Called before the row goes into ec following example uses this event or if the edit should be cancelled:
		
		`public onAutoGridBeforeEd Service.Communication.Eve BeforeEditEventData<TGrid processEvent: (processIt? if(this.NgData.commod == "API-6876")`
		
		`{ eventData.eventPr var testModal = { title: "A messageTe messageTy confirmBu closeButtonText: "No", closeClic { proce }, confirmCl { //Whe selection back on row. if(ev t (eventData.dataRow.uid); } } this.launchQModa`

		} }
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 for single-row mode on an existing row and when row, it fires after the initialize call t dataRow) to the row initialized with
onAutoGridEditButtonClick(eventData: EventData.AutoGrid.ButtonClickEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row	Called when the Edit button is click row into Edit mode. TODO: Need to support multiple r
onAutoGridCancelButtonClick(eventData: EventData.AutoGrid.CancelButtonClickEventData)	**eventData.gridId**: the ID of the auto grid	Called when the Cancel button is
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	Called before a new row was adde

	has taken 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 auto grid update update operation). TODO: Need to support multiple r
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 grid update su succeeded or not. Fired after data TODO: Need to support multiple r

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 auto grid delete s TODO: Need to support multiple r
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 grid delete su succeeded or not. Fired after data
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 grid Details l attempted. To override the default link naviga code, and be sure to set **eventDat** the default navigation.
onAutoGridDetailsLinkClose(eventData: EventData.AutoGrid.DetailsLinkCloseEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid	Called when the auto grid Details l The default behavior after closing refresh. If for some reason you wa

	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=true** (e.g. if you in the event handler (master data duplicate refresh of the calling gri For example, if you wish to refresh grid that called the details screen), **`eventData.eventProcessed this.ViewController.refre all external entity data`** If you wish to refresh the master fo **`eventData.eventProcessed this.ViewController.reque master data and all inter grids`**
onAutoGridAfterInit(eventData: EventData.AutoGrid.AfterInitEventData)	**eventData.gridId**: the ID of the auto grid	Called after the auto grid has beer

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 grid toolbar bu to enabled or disabled. Presen change. The purpose of this ev customize the enabled/disable in the selected grid row. For ex** `public onAutoGridToolbarS Service.Communication.Ev ToolbarSetStateEventData< void { if(eventData.dataR == "API-6053") eventData.buttor }` **WARNING: Do not do any othe handler. Only use the pattern a a dataRow value and setting th can potentially be called a lot s**
onAutoGridSelectionStateChange(eventData: EventData.AutoGrid.SelectionStateChangeEventData)	**eventData.gridId**: the ID of the auto grid **eventData.hasSelectedRows**: boolean, true if the grid has at least one row selected, false if no rows selected.	This event is only published for mu "selectionColumn". This event is o no rows selected to at least one ro for this is to enable/disable a form 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 button is click can be added using "addGridButto TODO: Need to support multiple ro
onAutoGridSchemaDefined(eventData: EventData.AutoGrid.SchemaDefinedEventData)	**eventData.gridId**: the ID of the auto grid **eventData.gridSchema**: grid schema object	Called when the grid schema is as
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. **eventData.action**:	Called when a create, update, or d

	"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 the field schem with dynamic column.

Properties/Functions

Set Field Value in Grid Row

ViewGrid.setRowFieldValue can be used to set values in grids:

public setRowFieldValue (rowData, fieldId: string, value: any, withoutRefresh = false): boolean

parameters:

- rowData: data row object to set field value on
- fieldId: id of the control within the grid
- value: value to set the field to
- withoutRefresh (optional, default to false): if true, sets the field value without doing a kendo refresh. For hierarchical grids, data should be updated without refreshing, otherwise kendo * will reload the whole grid.

return value: true if change was successful.

example:

```
public onAutoGridFieldChangeEvent(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
FieldChangeEventData<DTO.SuppInvoiceBankObservableObject>, processEvent: (processIt?: boolean) =>
void): void {
        // This is a hierarchical grid example - so skipping refresh
        if (eventData.fieldName == "payFormatGroupCode" && eventData.fieldValue ==
"updateParentField"){
                this.ViewGrid.setRowFieldValue(eventData.parentDataRows[0].rowData,"
payFormatTypePayInstrument","ParentFieldUpdated", true);
        }
}
```

TODO: add example how to get a specific grid row

For dates fields, make sure the value is a date object. See: Set Grid Date Value

Set Value on current row

Use ViewGrid.setCurrentRowFieldValue:

public setCurrentRowFieldValue (fieldId: string, value: any): boolean

parameters:

- fieldId: id of the field within the grid
- value: value to set the field to

return value: true if change was successful.

example to set the value of a boolean field in a grid:

```
// for boolean fields, value parameter should be True / False.
let grid=this.ViewController.getViewGrid("MyGrid");
grid.setCurrentRowFieldValue("MyField1",true);
```

For dates fields, make sure the value is a date object. See: Set Grid Date Value

Set Grid Date Value

When setting a value to a date field, you need to make sure that the value is a date object. Most dates on the UI are string values, in the format of the current culture. So, make sure you convert that string to a date object.

```
//dateValue is a string representing a date and this value needs to be in the format of the
current culture.
//For example: "3/21/2016" for en_US or "21/3/2016" for en_AU
let grid=this.ViewController.getViewGrid("MyGrid");
grid.setCurrentRowFieldValue("MyField1",kendo.parseDate(dateValue));
```

Set Field Value in Multiple Grid Rows

Use ViewGrid.setFieldValueInRows to set field values for multiple rows in a grid:

public setFieldValueInRows (fieldId: string, value: any, selectionFunction?: (rowData: kendo.data.Model) => boolean): void
{

Hide/Show Button in Grid Toolbar

parameters:

- fieldId: id of the field
- value: value to set the field to
- selectionFunction (optional): 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.

example 1: set MyField to value 123 for all rows:

```
let grid=this.ViewController.getViewGrid("MyGrid");
grid.setFieldValueInRows("MyField",123);
```

example 2: set MyField to value 123 for all rows where field isRestricted is set to true:

```
let grid=this.ViewController.getViewGrid("MyGrid");
grid.setFieldValueInRows("MyField",(rowData)=>{return rowData["isRestricted"]});
```

Disable/Enable Field in Grid Row

Use ViewGrid.setGridControlDisabled to enable or disable a grid field:

public setGridControlDisabled (fieldId: string, isDisabled: boolean)

parameters:

- fieldId: id of the control within the grid
- isDisabled: boolean indicating whether field should be set to disabled or made editable.

example:

```
let grid=this.ViewController.getViewGrid("MyGrid");
grid.setGridControlDisabled("MyField",true);
```

Add buttons to the Grid Toolbar

Use $scope.addGridButton for this:

public addGridButton(gridName: string, buttonName: string, text: string, cssClass?: string, htmlText?: string, insertHtmlTextAfter?: boolean, shortcut?: string)

* 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.

parameters:

- gridName: name (ID) of the grid
- buttonName: name of the button
- text: label text, should be already translated
- cssClass (optional): CSS class string to control the button color
- htmlText (optional): html text e.g. '' - insert icon
- insertHtmlTextAfter (optional): true - inserts html text after button text.

example:

- shows how to add a button to the Grid Toolbar.
- shows how html text can be added, e.g. display icon/ image in button before or after text.


```
public onBindData(eventData: Qad.Common.Service.Communication.EventData.QraView.
BindDataEventData<DTO.CountryMaint>): void {
    /**
```

```
     * Adds a button to the grid toolbar, the button is added to the end of the grid toolbar so
it will appear to the right of existing buttons.
     * parameters: gridName (string) - name of the grid
     *             buttonName (string)- name of the button
     *             text (string)- label text, should be already translated
     *             cssClass (string optional)- CSS class string to control the button color
     *             htmlText (string optional)- html text e.g. '<span class="fa fa-edit"></span>'
- insert icon
     *             insertHtmlTextAfter (boolean optional)- true - inserts html text after button
text.
     *             shortcut (string optional) FD28 - shortcut key combination (e.g. "alt+b").
     *                      IMPORTANT: IF USING A SHORCUT KEY FOR THE BUTTON THEN UX MUST FIRST
BE CONSULTED TO PREVENT KEY OVERLAP AND
     *                                 TO UPDATE HELP DOCUMENTATION WITH THE NEW SHORTCUT.
     */

    let myGrid=this.ViewController.getViewGrid("countryExtNOneToManyAutoGrid");

    // Add button with text
    this.$scope.addGridButton(myGrid.GridID, myGrid.GridID + "_NextLevel", "Next Level");

    // Add button with icon displayed to left hand side of button text.
    this.$scope.addGridButton(myGrid.GridID, myGrid.GridID + "_CustomEdit", "Custom Edit", "",
'<span class="fa fa-edit"></span>', false);

    // Add button with icon displayed to right hand side of button text.
    this.$scope.addGridButton(myGrid.GridID, myGrid.GridID + "_CustomAttach", "Custom Attach",
"", '<span class="fa fa-paperclip"></span>', true);

    // Add button with a shortcut key (FD28).
    this.$scope.addGridButton(myGrid.GridID, myGrid.GridID + "_ShortcutTest", "Shortcut Test",
null, null, false, "alt+b");
}

...


// Handle click event on button created above.
public onAutoGridButtonClick(eventData: EventData.AutoGrid.ButtonClickEventData<DTO.CountryExtN &
kendo.data.ObservableObject>): void {
    if (eventData.buttonName == this.ViewGrid.GridID +  "_NextLevel" ) {
        // Code to handle click event.
        alert("Grid:" + eventData.gridId + " Button :" + eventData.buttonName + " Clicked.");
    }
}
```

When using a shortcut key for the button, make sure it does not overlap with an existing shortcut anywhere on the screen.

Enable/Disable Button in Grid Toolbar

Use global setControlDisabled function:

setControlDisabled (elementId, ngElement, isDisabled) {

parameters:

- elementId: id of the element
- startElement: HTML element under which to find the field (i.e. this.$element)
- isDisabled: boolean indicating whether field should be set to disabled or made editable.

example:

```
let myGrid=this.ViewController.getViewGrid("countryExtNOneToManyAutoGrid");
setControlDisabled(myGrid.GridID + "_NextLevel", this.$element, true);
```

Hide/Show Button in Grid Toolbar

Use global setControlVisible function:

setControlVisible (elementId, ngElement, isDisabled) {

parameters:

- elementId: id of the element
- startElement: HTML element under which to find the field (i.e. this.$element)
- isDisabled: boolean indicating whether field should be set to disabled or made editable.

example:

```
let myGrid=this.ViewController.getViewGrid("countryExtNOneToManyAutoGrid");
setControlVisible(myGrid.GridID + "_NextLevel", this.$element, true);
```

Change Label of Custom Button in Grid Toolbar

Changing the label of a grid button needs to be done with JQuery as follows:

```
let myGrid=this.ViewController.getViewGrid("countryExtNOneToManyAutoGrid");
$("#"+myGrid.GridID + "_NextLevel",this.$element).text(escapeHtml("MyLabel"));
```

Force Save of Grid Row

This example forces a row to be dirty:

```
onAutoGridBeforeUpdate(eventData: EventData.AutoGrid.BeforUpdateEventData<DTO.CountryExtN & kendo.
data.ObservableObject>, processEvent: (processIt?: boolean) => void): void {
    (<any>eventData.dataRow).dirty=true;
}
```

Find Grid and Field of Row

To find the grid that a particular dataRow belongs to:

```
let grid=this.ViewController.getViewGrid($("[data-uid=" + eventData.dataRow.uid + "]",
this.$element).closest("[kendo-grid]").attr("id").substr(6));
```

Customize the Grid data URLs

Grid data url's can be changed in the onAutoGridBeforeInit event. Note that this event handler is implemented on the main view controller (see Main View). This is because this event is fired before the ViewGrid even exists.

Customize the Grid Initialize URL

```
public onAutoGridBeforeInit(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
BeforeInitEventData) {
{
        if(eventData.gridId == "AttributeCategoryLinkAutoGrid")
            eventData.dataWiring.initBaseUrl = "myCustomUrl" + &attributeName=
{attributeDefinitions[0].attributeName}";
}
```

Customize the Grid Details Link URL

```
if (eventType == QraEventService.autoGridEvents.beforeInit) {
          if(eventData.gridId == "TestEntityBAutoGrid")
                  eventData.dataWiring.linkUrl += "&keyC={keyC}";
}
```

Add a filter to the Grid list (read) URL

```
public onAutoGridBeforeInit(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
BeforeInitEventData) {
{
        if(eventData.gridId == "CommodityCodeDetailAutoGrid")
       eventData.dataWiring.listBaseUrl = addParamToUrl(eventData.dataWiring.listBaseUrl,
"filter=" + encodeURIComponent("kind,eq,WO,literal"));
}
```

Disable Field in Multi-row-edit Grid

Disable a field in a multi-row-edit grid for both New and Edit:

```
onAutoGridNewButtonClick(eventData: EventData.AutoGrid.ButtonClickEventData<DTO.CountryVatFormat
& kendo.data.ObservableObject>): void {
    this.ViewGrid.setGridControlDisabled ("taxFormat", true );
}

onAutoGridRowChange(eventData: EventData.AutoGrid.RowChangeEventData<DTO.CountryVatFormat & kendo.
data.ObservableObject>): void {
    this.ViewGrid.setGridControlDisabled ("taxFormat", true );
}
```

Disable Details Link When No Rows in Grid

Disable the Details link for a grid if there are no rows displayed in the grid. Because the Details link is automatically enabled by the Web UI framework on the dataBound event, no application JavaScript code is required to do this. Handlers like the example below are needed only to disable the link for the relevant special cases:

```
onAutoGridBindData(eventData: EventData.AutoGrid.BindDataEventData<DTO.CountryExtN & kendo.data.
ObservableObject>): void {
    if (eventData.dataRows.length == 0) {
       this.ViewGrid.DetailsLinkEnabled=false;
    }
}
```

Disable Details Link Based On Parent Value

Disable the Details link for a grid based on the value of a field in the parent form. Because the Details link is automatically enabled by the Web UI framework on the dataBound event, no application JavaScript code is required to do this. Handlers like the example below are needed only to disable the link for the relevant special cases:

```
onAutoGridBindData(eventData: EventData.AutoGrid.BindDataEventData<DTO.CountryExtN & kendo.data.
ObservableObject>): void {
    // Disable Details link if CHARACTER FIELD on form contains the text "NODETAILS"
    if (this.ViewController.getViewField("Field2AutoField1").Value=="NODETAILS") {
        this.ViewGrid.DetailsLinkEnabled=false;
    } else {
        this.ViewGrid.DetailsLinkEnabled=true;
    }
}
```

Modify Multi-row-edit Grid Datasource

Programmatically modify the contents of a multi-row-edit grid.

The array of data under this.NgData that holds the rows of a multi-row-edit grid can be considered the master data source for the multi-row-edit grid. The Framework populates the grid from that data. However, the grid keeps its own datasource which is a copy of that data and the Framework synchronizes the two at various points.
The most efficient way to programmatically modify multi-row-edit grid data is to work with the NgData source. In other words, do the necessary deletions and additions in the NgData source array, and then call the framework to sync that source with the grid.

Below is an example where a row of data is added and row deleted from a multi-row-edit grid. Note: this assumes the underlying datatype on the server is DATE or DATETIME. If DATETIME-TZ is used, then convert the DATETIME-TZ variable to a UTCString.

```
onFieldChange(viewField: IViewField<any>, eventData: EventData.QraView.FieldChangeEventData<any>,
processEvent: (processIt?: boolean) => void): void {
    if (viewField.Name=="GlThisLevelTotalAutoField") {
        if(eventData.fieldValue)
        {
            //Make a copy of the first data row from the grid's ngData datasource
            var firstDataRow = this.NgData.tableAs[0].childTableA1.grandchildTableA11s[0];
            var copyFirstDataRow = JSON.parse(stringifyIgnoreTimezone(firstDataRow));

            //Make a change to the data in that copied row
            copyFirstDataRow.field3 = "v4";

            //Add that copied row to the ngData array that represents the grid data
            this.NgData.tableAs[0].childTableA1.grandchildTableA11s.push(copyFirstDataRow);

            //Remove the second data row in the ngData array that represents the grid data
            this.NgData.tableAs[0].childTableA1.grandchildTableA11s.splice(1,1);

            //Refresh the grid
            this.ViewController.getViewGrid("GrandchildTableA11AutoGrid").refreshDataGrid(false);
        }
    }
}
```

Fixed width grids - Add a class "qIgnoreResize", so resize function ignores it

Use ViewGrid.KendoGrid.element to modify html:

```
let myGrid=this.ViewController.getViewGrid("countryExtNOneToManyAutoGrid");
//Set Fixed Width for Data Grid
myGrid.KendoGrid.element.width(200);
//Add "qIgnoreResize" class, so resize function will ignore it.
myGrid.KendoGrid.element.addClass("qIgnoreResize");
```

Disable/Enable a grid while still allowing grid interaction

Use ViewGrid.setGridDisabled utility

This utility will disable the user from editing/deleting/adding records to the grid. They will be able to sort/re-arrange columns, click on rows, and with hierarchical grids open child grids. The utility has two parameters, whether to enable or disable, and whether it should disable all child grids - (disable:boolean, affectChildLevels:boolean). The second parameter is optional and should only be used for hierarchical grids.

example:

```
//Disable only this grid (second parameter is optional, and only needs to be used for
hierarchical grids)
if (this.NgData && this.NgData.supplierInvoices[0].suppInvoiceReference == "disableGridTest"){
        this.suppInvoiceBankAutoGridTSHandler.ViewGrid.setGridDisabled(true, false);
}


//Enable only this grid (second parameter is optional, and only needs to be used for hierarchical
grids)
if (this.NgData && this.NgData.supplierInvoices[0].suppInvoiceReference == "enableGridTest"){
        this.suppInvoiceBankAutoGridTSHandler.ViewGrid.setGridDisabled(false, false);
}


//For hierachical grids


//Disable ALL grids below and including this grid
if (this.NgData && this.NgData.supplierInvoices[0].suppInvoiceReference == "disableGridTestAll"){
        this.suppInvoiceBankAutoGridTSHandler.ViewGrid.setGridDisabled(true, true);
}


//Enable ALL grids under and including this grid
if (this.NgData && this.NgData.supplierInvoices[0].suppInvoiceReference == "enableGridTestAll"){
        this.suppInvoiceBankAutoGridTSHandler.ViewGrid.setGridDisabled(false, true);
}
```

Enable and tab to a field that is disabled in a Multi-row-edit grid

In this example, a row in a multi-row-edit grid has only one field enabled (salespersonCode). When tabbing out of that field, we want to enable the commissionPercent field and focus on it.
NOTE: this is only necessary because salespersonCode is the only enabled field in the row so when it is tabbed out, the row closes. This prevents the row from closing. The code below is not needed if there are other enabled fields in the row.

```
onAutoGridFieldChangeEvent(eventData: EventData.AutoGrid.FieldChangeEventData<any>, processEvent:
(processIt?: boolean) => void): void {
    if (eventData.fieldName=="GlThisLevelTotalAutoField") {
        if(eventData.fieldValue && eventData.fieldValue.trim()) { //Non-blank value of
salespersonCode
            selectMultiGridRow(this.$element, this.ViewGrid.GridID, eventData.dataRow.uid);
            this.ViewGrid.setGridControlDisabled("commissionPercent",false);
            var commissionField = $("#commissionPercent", this.ViewGrid.KendoGrid.element);
            setFocusToField(commissionField);
        }
        else {
            this.ViewGrid.setGridControlDisabled("commissionPercent",true);
        }
    }
}
```

Prevent the autoGridEvents.dataBound event

Prevent the autoGridEvents.dataBound event from occurring when setting data on a grid. This function is usually used when the data is bound in the event handler instead of in the framework.

Use preventGridDataBound global function for this:

preventGridDataBound(kGrid:any, fn: function)

parameters:

- kGrid: kendo grid object: can be retrieved with ViewGrid.KendoGrid property
- fn: function to call for binding the grid data (function with no parameters)

example:

```
preventGridDataBound(this.ViewGrid.KendoGrid, ()=> {
     //Set grid data here.
});
```

Forcing grids to refresh when saving an existing record

For efficiency the framework does not refresh single-row grids when an existing header record is saved.

In some cases, it might be desirable for a grid to refresh at this point, for example, if saving the header updates values of the grid lines.

To force a grid to refresh in this case, add the following to the js-handler code:

```
public onAutoGridBeforeInit(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
BeforeInitEventData) {
   if (eventData.gridId == "ItemSiteInventoryAutoGrid")
      eventData.viewSchema.gridSettings.alwaysRefresh = true;
}
```

Posting grid date values to the server

The Kendo grid works with date values as JavaScript Date objects. For example, a date value in the grid's underlying data source is a JavaScript Date object with a value like this:
"Tue May 31 2016 17:00:00 GMT-0700 (Pacific Daylight Time)"

When posting grid data to the server, it is necessary to convert the grid data to JSON. If JSON.stringify() is used, then by default it will convert any date values to UTC format (e.g. "2016-05-31T00:00:00.000Z"). This is good. And in doing this it also converts the date value to UTC (This is **not** good). We do **not** want to convert the value. Therefore, if handler code needs to post grid data to the server that contains date values, then it should use the utility stringifyIgnoreTimezone() and should NOT use JSON.stringify(). stringifyIgnoreTimezone() will **not** do any date value conversions.

A better way is to "stringify" while you still have the metadata describing the target field. The grid has the knowledge of what is expected.

Use of stringIfyCorrectTimezone

```
onAutoGridBeforeEdit(eventData: EventData.AutoGrid.BeforeEditEventData<DTO.CountryVatFormat &
kendo.data.ObservableObject>, processEvent: (processIt?: boolean) => void): void {
    // Account for expected data type on server
    var gridRowData = (<Qad.QraView.UI.ViewGrid><any>this.ViewGrid).correctGridTimezone(this.
ViewGrid.getSelectedRowData());
    eventData.eventProcessed=true;
}
```

Disable/Enable a Grid Column

This example shows how to disable a column(s) in a grid when a checkbox is selected, and enable it when it is clear.

"TestEntityBAutoGrid" is the name of the grid, and "field5" is the data-field (to disable a column in a KendoGrid, the data-field name is needed because that is how Kendo uniquely identifies columns).

```
/*
 * setGridColumnDisabled: disables/ enables the specified column in a grid based on field
 *
 * fieldId: field id(s) of the column field. Pass a string or array of string.
 * isDisabled: boolean indicating whether column field should be set to disabled or made editable.
 *            isDisabled is true --> disable
 *            isDisabled is false --> enable or editable
 *
 */
public setGridColumnDisabled (fieldId: string | string[], isDisabled: boolean)

/* Sample TS Handler example */
public onAutoGridBindData(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
BindDataEventData<DTO.TestEntityB>): void{
        if (this.NgData && this.NgData.tableAs[0].keyA == "A-1" && this.NgData.tableAs[0].field2
== "API16728_1")
                this.ViewGrid.setGridColumnDisabled(["field4","field8"], true); /* Use array of
field names to improve performance. */
        else if (this.NgData && this.NgData.tableAs[0].keyA == "A-1" && this.NgData.tableAs[0].
field2 == "API16728_2") {
                this.ViewGrid.setGridColumnDisabled("field4", true);
                this.ViewGrid.setGridColumnDisabled("field8", false);
        }
        else if (this.NgData && this.NgData.tableAs[0].keyA == "A-1" && this.NgData.tableAs[0].
field2 == "API16728_3"){
                this.ViewGrid.setGridColumnDisabled("field4", false);
                this.ViewGrid.setGridColumnDisabled("field8", true);
        }
        else
                this.ViewGrid.setGridColumnDisabled(["field4","field8"], false);
}
```

Hide/Show a Grid Column

This example shows how to hide a column in a grid when a checkbox is selected, and show it when it is clear.

"TestEntityBAutoGrid" is the name of the grid, and "field5" is the data-field (to hide a column in a KendoGrid, the data-field name is needed because that is how Kendo uniquely identifies columns).

Note: it will uncheck the field in the configured panel.

```
onAutoGridFieldChangeEvent(eventData: EventData.AutoGrid.FieldChangeEventData<any>, processEvent:
(processIt?: boolean) => void): void {
    if (eventData.fieldName == "Field4AutoField"){
        if (eventData.fieldValue){
            this.ViewGrid.hideColumn("field5",false);
        }
        else{
            this.ViewGrid.hideColumn("field5",true);
        }
```

```
    }

}
```

Set Focus to Field in Grid

This example shows setting focus to a field in a grid.

Single-row edit grids will only set focus to the field if the rowUID is the row already in edit mode, multi-edit grids will select the row and put focus in the desired field.

Function requires the name of the grid, the rowUID, and the field name ("field5").

```
//Single row edit grid
onAutoGridBeforeEdit(eventData: EventData.AutoGrid.BeforeEditEventData<DTO.CountryVatFormat &
kendo.data.ObservableObject>, processEvent: (processIt?: boolean) => void): void {
    var rowUID = eventData.dataRow.uid;
    this.ViewGrid.setFocusToGridField(rowUID, "field5");
}

//Multi row edit grid - this example gets the rowUID from the first row in the grid
onAutoGridBeforeEdit(eventData: EventData.AutoGrid.BeforeEditEventData<DTO.CountryVatFormat &
kendo.data.ObservableObject>, processEvent: (processIt?: boolean) => void): void {
   var gridData = this.NgData.tableAs[0].childTableA2s;
   if (gridData.length > 0){
        var row = gridData[0];
        var rowUID = row.uid;
        this.ViewController.getViewGrid("ChildTableA2AutoGrid").setFocusToGridField(rowUID,
"field5");
    }
}
```

Data Grid - Building Grid Error Context

Build the context for a grid error. If passing in optional rowData and fieldName, it will build the error context for the grid to that point.

this.ViewGrid.buildGridErrorContext(rowData?, fieldName?)
param rowData – If rowData is passed in, it will add the row information.
param fieldName – If rowData and fieldName is passed in, it will go to the field.

```
public onAutoGridFieldChangeEvent(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
FieldChangeEventData<DTO.ChildTableA21ObservableObject>, processEvent: (processIt?: boolean) =>
void): void {     if (eventData.fieldName == "field2"){
               if (eventData.fieldValue == "error"){
                       let gridErrors = [];
                        gridErrors.push(new Error({message: "Grid Error Test", fieldName:
eventData.fieldName, context: this.ViewGrid.buildGridErrorContext(eventData.dataRow, "field2")}));
                        this.ViewController.ErrorGroupPanel.addErrorsToErrorGrid(gridErrors);
                        this.ViewController.ErrorGroupPanel.showErrorGrid();
                }
        }
}
```

Change Key Fields Used for Client Side Duplicate Grid Rows Check

To change the key fields that are used for the duplicate grid row check, use setKeyFieldsForDuplicateCheck.

```
public onAutoGridBindData(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
BindDataEventData<DTO>): void{
        //Pass in a comma separated list of keys that need to be used for the client side
duplicate row check
        this.ViewGrid.setKeyFieldsForDuplicateCheck("entityCode,suppInvoiceRegistrationNr,
bankNumber,bankNumberExtension");
}
```

Hierarchical Grids

Get and Set ViewGridData on ViewGrid

Because of the complexity of updating/changing data in NgData in hierarchical grids, the ViewGridData will have the most up to date data for the grids. In order to change or access this data, two functions have been added to ViewGrid.

Implement this handler in your ViewGridTSHandler class:

```
public onAutoGridAfterInit(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
AfterInitEventData): void{
   if (eventData.parentDataRows[0].rowData["bankNumberValidation"] == "13031"){
      if (this.ViewGrid.getViewGridData("[0].suppInvoiceBankPayCodes")).length == 0){
          // params: path - the json path of the data you want to set
          // data: the data you want to put at the path specified
          this.ViewGrid.setViewGridData("[0].suppInvoiceBankPayCodes",data);
      }
   }
}
```

Hierarchical Grids - Hide Toggle Icons

The toggle icon is used to open child grids on hiearchical grids.

Two functions were created in ViewGrid which can be called from the ViewGridTSHandler for hiding these icons so a child grid can't be opened.

UX must be consulted before using these functions and changing the default behavior.

```
// This hides all toggle icons on rows that currently do not have any children
// This will only work on internal (multi-edit) grids since we do not know child information for
external(single edit)
// Again, consult with UX before using this
// @param hide, whether to hide (if true) or show (if false) the toggle icon that is used to open
child grids
// @param disableAddChild, whether to also disable the add child button for grid rows with a
hidden toggle icon
public onAutoGridBindData(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
BindDataEventData<DTO.SuppInvoiceBank>){
    // For example, this is hiding the toggle icon and disabling add child for rows with toggle
icon
    this.ViewGrid.hideEmptyChildGrids(true,true);
}


// This hides all toggle icons for the rows in the passed in JQuery selected
// @param rows, a JQuery selection of rows that the icon should be hidden on
// @param hide, whether to hide (if true) or show (if false) the toggle icon that is used to open
child grids
// @param disableAddChild, whether to also disable the add child button for grid rows with a
hidden toggle icon
public onAutoGridBindData(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
BindDataEventData<DTO.SuppInvoiceBank>){
    // For example, this is hiding the toggle icons of all rows with the k-alt styling
    var hideSomeToggleIcons = $(".k-alt", this.$element);
    this.ViewGrid.hideRowsToggleIcon(hideSomeToggleIcons,true,true);
}
```

Hierarchical Grids - Put ViewGridData into NgData before sending (bindMultiGridToModel)

In Internal Hierarchical Grids, the dataSource that is maintained while users are editing is ViewGridData. However, what is sent is NgData. If you are sending NgData yourself and not through the foundation save (such as with a pop-up window or own ajax post), then before sending, bindMultiGridToModel should be used to bind all of the ViewGridData changes to NgData.

```
public onButtonClick(viewButton: IViewButton, eventData: Qad.Common.Service.Communication.
EventData.QraView.ButtonClickEventData): void {
        if (eventData.buttonId === "ToolBtnSubmit") {
                this.$scope.bindMultiGridsToModel(); // if hierarchical grid, bind the
ViewGridData to NgData
                var dataToPost = this.NgData;
```

```
        var targetUrl = "api/acme/items?domainCode=" + encodeURIComponent(this.NgData.
commodityCodeDetails[0].domainCode) +
                    "&itemCode=" + encodeURIComponent(eventData.fieldValue);
```

Data Grid/Multi-child Hierarchical Grids - setViewGridOptions

Helper method used to set grid/row data options. An example of its usage may be found in JournalEntryPostingLinesAutoGridTSHandler.ts.

```
         /**
         * setViewGridOptions : Set View Grid options
         * @param dataRow: data Row. If dataRow is null, applies to grid.
         * @param viewGridOptions: json object : contains Name(s) of property and their values.
         * Options Supported:
         *  expanded: Set Grid row attribute "expanded" to true/false, so refresh will auto
expands row.
         *  tabSettings: array of digit 0 or 1. 0-hidden 1-Visible
         *  defaultDisplayLines: Set default Display Lines setting of the grid, This sets value
and resizes grid.
         */
        setViewGridOptions(dataRow: any, viewGridOptions: any);

                //Example: below example tabSettings arrar values: 1-Visible 0-Hidden
                viewGrid.setViewGridOptions(dataRow, { "expanded": true, "tabSettings": [1,1,0,0]
});


                //Example: below example sets default Display Lines setting of the grid
        this.ViewGrid.setViewGridOptions(null, {defaultDisplayLines: 50 });
```

Data Grid/Multi-child Hierarchical Grids - refreshDataGridRow

Helper method used to refresh datagrid row including its child data grids & forms. An example of its usage may be found in JournalEntryPostingLinesAutoGridTSHandler.ts.

```
         /**
         * refreshDataGridRow: Refreshes datagrid row including it's child views.
         * @param rowData - JSON data row, must include "uid" to find html row element from grid.
         * @param refreshOptions - refresh options
                 *     keepInEdit - default true - dictates whether the refresh should keep the
row in edit mode or remove the edit state
                 *     foceSync - default true - for performance reasons, can put off full grid
sync during refresh (should be used when keepInEdit is true because sync will occur on save of
row)
                 *     refreshChildGrids - default true - can optionally stop the full refresh of
the grids children, can improve performance if refresh of child data isn't necessary
         */
        public refreshDataGridRow (rowData: any, refreshOptions?: any)
```

Refresh/repaint values on a row without a Kendo refresh

Doing a refresh or a set on a row can be bad for performance. To get around this performance hit, you can use kendoFastReDrawRow after setting the values on the row.

Note: this will not update aggregates/grouping for the new data. However, this isn't usually updated for every change in a field in a row - it will be updated on save of the row. Save happens when focus leaves the row for internal grids, and when clicking done or next line on an external grid.

```
public onAutoGridFieldChangeEvent(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
FieldChangeEventData<DTO>, processEvent: (processIt?: boolean) => void): void {
        let selectedRow = this.ViewGrid.SelectedRow;
        let dataItem = this.ViewGrid.dataItem(selectedRow);
        for (var field in line) {
                dataItem[field] = line[field];
        }
        this.ViewGrid.kendoFastReDrawRow(this.ViewGrid.KendoGrid,selectedRow);
}
```

Multi-Child Hierarchical Grids

Multi-child Hierarchical Grids - childInit

Event published when a hierarchical child is created.

```
/**
 * Called when hierarchical child(s) are created
 */
public onAutoGridChildInit(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
ChildStateEventData<kendo.data.ObservableObject>): void {
    for (var i=0;i<this._iViewGridTSHandlerEvents.length;i++){
        this._iViewGridTSHandlerEvents[i].onAutoGridChildInit(eventData);
    }
}
```

Multi-child Hierarchical Grids - childCollapse

Event published when a hierarchical child is collapsed.

```
/**
 * Called when hierarchical grid row collapsed
 */
public onAutoGridChildCollapse(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
ChildStateEventData<kendo.data.ObservableObject>): void {
    for (var i=0;i<this._iViewGridTSHandlerEvents.length;i++){
        this._iViewGridTSHandlerEvents[i].onAutoGridChildCollapse(eventData);
    }
}
```

Multi-child Hierarchical Grids - childExpand

Event published when a hierarchical child is expanded.

```
/**
 * Called when hierarchical grid row expanded
 */
public onAutoGridChildExpand(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
ChildStateEventData<kendo.data.ObservableObject>): void {
    for (var i=0;i<this._iViewGridTSHandlerEvents.length;i++){
        this._iViewGridTSHandlerEvents[i].onAutoGridChildExpand(eventData);
    }
}
```

Multi-child Hierarchical Grids - childActivate

Event published when a child tab is activated in tabbed hierarchical grid.

```
/**
 * Called when hierarchical grid child is activated.
 */
public onAutoGridChildActivate(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.
ChildStateEventDataV2<DTO.JournalEntryObservableObject>): void {
        if (eventData.additionalData.childName == "DataTypeTestAutoGrid") {
                this.ViewGrid.setChildViewGridFormFieldVisible("DataTypeTestAutoGrid", eventData.
dataRow, "DecimalFieldAutoField", (eventData.dataRow.glAccount != "1090"));
                this.ViewGrid.setChildViewGridFormFieldDisabled("DataTypeTestAutoGrid", eventData.
dataRow, "DataListFieldAutoField", (eventData.dataRow.glAccount = "1090"));
        }
}
```

Multi-child Hierarchical Grids - childFormFieldChange

Event published when a hierarchical child is a form view type and one of its fields change.

```
/*
 * Called when hierarchical grid form field value is changed.
 */
public onAutoGridChildFormFieldChange(eventData: Qad.Common.Service.Communication.EventData.
AutoGrid.ChildFormFieldEventData<kendo.data.ObservableObject>): void {
    for ( var i = 0; i < this._iViewGridTSHandlerEvents.length; i++ ) {
        this._iViewGridTSHandlerEvents[i].onAutoGridChildFormFieldChange(eventData);
    }
}
```

Multi-child Hierarchical Grids - childFormFieldLeave

Event published when a hierarchical child is a form view type and focus leaves one of its fields.

```
/**
 * Called when hierarchical grid form field leave.
 */
public onAutoGridChildFormFieldLeave(eventData: Qad.Common.Service.Communication.EventData.
AutoGrid.ChildFormFieldEventData<kendo.data.ObservableObject>): void {
    for (var i=0;i<this._iViewGridTSHandlerEvents.length;i++){
        this._iViewGridTSHandlerEvents[i].onAutoGridChildFormFieldLeave(eventData);
    }
}
```

Multi-child Hierarchical Grids - setViewGridChildrenVisible

Helper method used to show/hide View Grid children (tabs).

```
        /**
         * setViewGridChildrenVisible : show/Hide View Grid child(s)
         * @param dataRow: Parent data Row of child Grids. If dataRow is null, applies to all
grid rows.
         * @param childGridsInfo: json object : contains Name(s) of child grid to show/hide &
isVisible flag.
         *        e.g. [{"name": "SuppInvoiceBankPayCodeAutoGrid", "isVisible": True}]
         */
        public setViewGridChildrenVisible(dataRow: any, childGridsInfo: any)


        /**
         * Sample Example of TSHanlder of ViewGrid.
         */
        public onAutoGridChildExpand(eventData: Qad.Common.Service.Communication.EventData.
AutoGrid.ChildStateEventData<DTO.JournalEntryObservableObject>): void {
            console.log("JournalEntryPostingLinesAutoGridTSHandler.onAutoGridChildExpand GridID:
" + eventData.gridId);
            this.ViewGrid.setViewGridChildrenVisible(eventData.dataRow, [{"name":
"AmountsAutoGrid", "isVisible": (eventData.dataRow.glAccount != "1090")}]);
        }
```

Multi-child Hierarchical Grids - setViewGridChildrenDisabled

Helper method used to disable/enable View Grid children (tabs).

```
        /**
         * setViewGridChildrenDisabled : Enable/Disable View Grid child(s)
         * @param dataRow: Parent data Row of child Grids. If dataRow is null, applies to all
grid rows.
         * @param childGridsInfo: json object : contains Name(s) of child grid to enable/disable
& isDisabled flag.
         *        e.g. [{"name": "SuppInvoiceBankPayCodeAutoGrid", "isDisabled": True}]
         */
        public setViewGridChildrenDisabled(dataRow: any, childGridsInfo: any)


        /**
         * Sample Example of TSHanlder of ViewGrid.
         */
```

```
        public onAutoGridChildExpand(eventData: Qad.Common.Service.Communication.EventData.
AutoGrid.ChildStateEventData<DTO.JournalEntryObservableObject>): void {
            console.log("JournalEntryPostingLinesAutoGridTSHandler.onAutoGridChildExpand GridID:
" + eventData.gridId);
             this.ViewGrid.setViewGridChildrenDisabled(eventData.dataRow, [
                 {"name": "AmountsAutoGrid", "isDisabled": (eventData.dataRow.glAccount ==
"1091")},
                 {"name": "SafAnalysisAutoGrid", "isDisabled": (eventData.dataRow.glAccount ==
"1092")},
                 {"name": "DataTypeTestAutoGrid", "isDisabled": (eventData.dataRow.glAccount ==
"1093")}]);

        }
```

Multi-child Hierarchical Grids - expandViewGridChildren

Helper method used to expand/collapse View Grid children.

```
        public onAutoGridFieldChangeEvent(eventData: Qad.Common.Service.Communication.EventData.
AutoGrid.FieldChangeEventData<DTO.JournalEntryObservableObject>, processEvent: (processIt?:
boolean) => void): void {
            // Expand children on leave of first field of the row.
            if (eventData.fieldName == "glAccount" && eventData.fieldValue != "") {
                this.ViewGrid.expandViewGridChildren(eventData.dataRow, true);
            }
            // Expand children on leave of last field of the row.
            if (eventData.fieldName == "transactionCurrency") {
                this.ViewGrid.expandViewGridChildren(eventData.dataRow, (eventData.fieldValue !=
"USD"));
            }
        }
```

Multi-child Hierarchical Grids - setChildViewGridFormFieldVisible

Helper method used to hide/show View Grid Form input fields.

```
/**
 * setChildViewGridFormFieldVisible: Hide/Show fields in child Forms.
 * @param gridFormName: (string) name of the grid form
 * @param dataRow: (json) parent data row
 * @param fieldNames: (array or string) name(s) of the field
 * @param isVisible: (boolean) Values: True-Show field, False-Hide field.
 * @param updateVisible: (boolean) Optional - True(Default) - updates isBLHiddenClient flag.
 */
setChildViewGridFormFieldVisible(gridFormName: string, dataRow: any, fieldNames: any, isVisible:
boolean, updateVisible?: boolean);

Example:
this.ViewGrid.setChildViewGridFormFieldVisible("DataTypeTestAutoGrid", eventData.dataRow,
"DecimalFieldAutoField", (eventData.dataRow.glAccount != "1090"));
```

Multi-child Hierarchical Grids - setChildViewGridFormFieldDisabled

Helper method used to disable/enable View Grid Form input fields.

```
/**
 * setChildViewGridFormFieldDisabled: Disable/Enable field in child Forms.
 * @param gridFormName: (string) name of the grid form
 * @param dataRow: (json) parent data row
 * @param fieldNames: (array or string) name(s) of the field
 * @param isDisabled: (boolean) Values: True-Disables field, False-Enables field.
 * @param updateDisabled: (boolean) Optional - True (Default) - updates disabled flag.
 */
public setChildViewGridFormFieldDisabled(gridFormName: string, dataRow: any, fieldNames: any,
isDisabled: boolean, updateDisabled = true)
```

```
Example:
public onAutoGridAfterInit(eventData:  Qad.Common.Service.Communication.EventData.AutoGrid.
AfterInitEventData): void {
   viewGrid.setChildViewGridFormFieldDisabled("PostingDetailsAutoGrid", eventData.dataRow,
               [PostingDetailsGridAutoFields.SUB_ACCOUNT_AUTO_FIELD,
               PostingDetailsGridAutoFields.COST_CENTER_AUTO_FIELD,
               PostingDetailsGridAutoFields.PROJECT_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_DEBIT_BC_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_DEBIT_SC_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_CREDIT_BC_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_CREDIT_SC_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_QTY_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_EXCHANGE_RATE_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_RATE_SCALE_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_CCRATE_AUTO_FIELD,
               PostingDetailsGridAutoFields.POSTING_LINE_CCSCALE_AUTO_FIELD,
               PostingGLOpenItemsGridAutoFields.ALLOCATION_KEY_AUTO_FIELD,
               PostingGLOpenItemsGridAutoFields.ALLOCATION_POSTING_LINE_AUTO_FIELD,
               PostingGLOpenItemsGridAutoFields.ALLOCATION_TYPE_AUTO_FIELD,
               PostingCrossCoGridAutoFields.CROSS_ENTITY_AUTO_FIELD,
               PostingCrossCoGridAutoFields.INTERCO_BR_AUTO_FIELD], true);
}
```

Multi-child Hierarchical Grids - setChildViewGridFormFieldValue

Helper method used to set View Grid Form input field's value.

```
/**
 * setChildViewGridFormFieldValue: Set field value in child Forms.
 * @param gridFormName: (string) name of the grid form
 * @param dataRow: (json) parent data row
 * @param fieldName: (string) name of the field
 * @param fieldValue: (any) - field value.
 */
setChildViewGridFormFieldValue(gridFormName: string, dataRow: any, fieldName: string, fieldValue:
any);


Example:
viewGrid.setChildViewGridFormFieldValue("PostingDetailsAutoGrid", postingLineGridRow,
"ExchangeRateAutoField", 9);
```

Group Panel (GroupPanelNavigator and GroupPanel)

Getting access to a GroupPanel object

Group panels are accessible through the GroupPanelNavigator property on the ViewController or via the event handler parameter:

1) Via ViewController:

```
let gps=this.ViewController.GroupPanelNavigator.getGroupPanelByName("MyGroupPanel");
```

2) As a parameter of an event handler:

```
onGroupPanelSelectorClick(groupPanel: Qad.QraView.TSHandler.IGroupPanel, eventData: EventData.
QraView.GroupPanelSelectorClickEventData): void {
   if (groupPanel.Name=="MyGroupPanel") {

   }
}
```

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.	**eventData.groupPanelId**: ID of group panel that was selected	Called when the error group panel's expansion toggle

QraView.GroupPanelToggleExpandEventData)	**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	control is clicked.
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.

Properties/Functions

Show/Hide group panel

The new way

IsVisible property

```
groupPanelObject.IsVisible = true;
groupPanelObject.IsVisible = false;
```

The old way

Use the GroupPanelNavigator.setGroupPanelVisible:

setGroupPanelVisible(panelName, isVisible, includeSubPanels, repositionNav, updateDisabled)

parameters:

- panelName, name of the panel (ID in the DOM)
- isVisible: boolean indicating whether panel should be visible or hidden.
- includeSubPanels, (optional) boolean that determines if the panel's children should also be shown. Defaults to true.
- repositionNav, (optional) boolean that determines if the navigator should reposition. If true, it will reposition to the current nav or first visible nav if the current one is hidden. Defaults to false.
- updateDisabled, boolean to disable checkbox for group panel and its child elements, so user can't hide/show them. If true, checkbox will be disabled even if panel is visible or hidden. If false, checkbox will not be disabled. Default value is true.

Note: Hiding a panel that has sub-panels will also hide those sub-panels.

example:

```
this.ViewController.GroupPanelNavigator.setGroupPanelVisible("MyGroupPanel",false);
```

Enable/Disable group panel

The new way

IsEnabled and EnableChildrenAlong properties

```
//It disables all controls of this panel and this panel` sub-panels
groupPanelObject.IsEnabled


//It disables all controls of only this panel
groupPanelObject.EnableChildrenAlong = false;
groupPanelObject.IsEnabled = false;
```

Almost all controls are enabled by groupPanelObject.IsEnabled = true, except for controls, which are disabled by business security, etc.

The old way

Use the utility function "setControlsDisabled" to disable group panels:

```
this.ViewController.setControlsDisabled($("#MyGroupPanel",this.$element),true);
```

Error viewer (ErrorViewer)

Getting access to the Error Viewer

The wrapper object for the error viewer is ErrorGroupPanel. It can be accessed in the following two ways:

1) Via ViewController:

```
this.ViewController.ErrorGroupPanel;
```

2) as a parameter of an event handler:

```
onErrorGroupPanelToggleExpand(errorGroupPanel: Qad.QraView.TSHandler.IErrorGroupPanel, eventData:
EventData.QraView.GroupPanelToggleExpandEventData): void {
    errorGroupPanel.removeFieldErrorsFromErrorGrid("MyField");
}
```

Events

Event Type	Event Data Properties	Description
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 **eventData.expand**: integer equal to 0 if the panel is being un-expanded, 1 if the panel is being expanded	Called when the error group panel's expansion toggle control is clicked.

Properties/Functions

Show/Hide error viewer

show error grid:

```
this.ViewController.ErrorGroupPanel.showErrorGrid();
```

hide error grid:

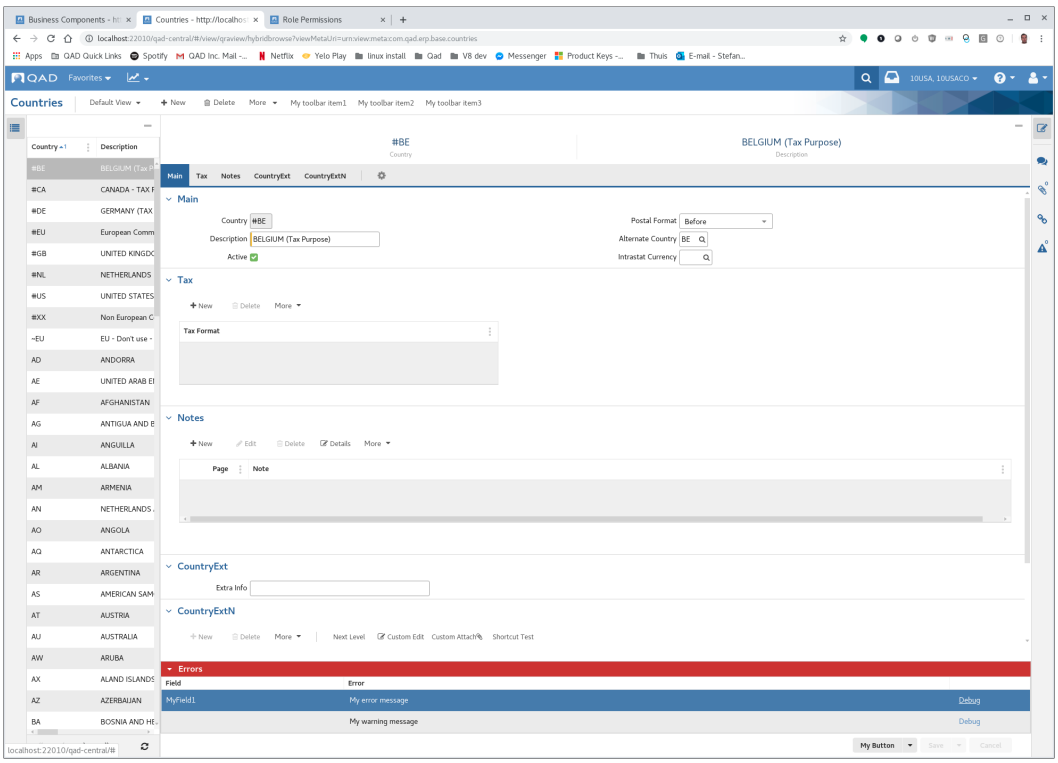
```
this.ViewController.ErrorGroupPanel.hideErrorGrid();
```

Add errors to error viewer

code:

```
let error=new Qad.Common.DTO.Error();
error.severity=Qad.Common.DTO.Error.SEVERITY_ERROR;
error.message="My error message";
error.fieldName="MyField1";
error.fieldValue="value1";
let warning=new Qad.Common.DTO.Error();
warning.severity=Qad.Common.DTO.Error.SEVERITY_WARNING;
warning.message="My warning message";
warning.fieldName="MyField2";
warning.fieldValue="value2";
this.ViewController.ErrorGroupPanel.addErrorsToErrorGrid([error,warning]);
this.ViewController.ErrorGroupPanel.showErrorGrid();
```

output to screen:



Clear error viewer

```
this.ViewController.ErrorGroupPanel.clearErrorGrid();
```

Summary Panel

Getting access to the Summary Panel

How to access the summary panel object

```
const summaryPanelObject = this.ViewController.GroupPanelNavigator.SummaryPanel;
```

Events

There are no events related to the summary panel.

Properties/Functions

Hide/Show the SummaryPanel

Hide/Show the SummaryPanel

```
summaryPanelObject.IsVisible = false;
```

Hide/Show a field on the Summary Panel (setSummaryPanelControlVisible)

The new way

Hide/Show fields on SummaryPanel

```
summaryPanelObject.hideField('NameAutoField1');
summaryPanelObject.showField('NameAutoField1');
```

The old way

Here is an example in which the TS handler code is showing or hiding the summary panel field.

```
        public onBindData(eventData: Qad.Common.Service.Communication.EventData.QraView.
BindDataEventData<any>) {
            if (eventData.data && this.NgData.approvals[0]) {
                // Set the Alternate Entity details for Attachments and Activity Feed to display
               The old way this.ViewController.setAlternateEntity(this.NgData.approvals[0].
entityInstanceUri, this.NgData.approvals[0].serviceProviderName, this.approvalUtil.
getAlternateKeyFields(this.NgData.approvals[0].alternateKeyFields), "approvals[0].
approvalObject");

                if (this.NgData.approvals[0].permittedDocumentUrl && this.NgData.approvals[0].
```

```
documentUrl != "") {
                    setSummaryPanelControlVisible(this.$scope, "ApprovalLinkAutoField",
this.$scope.ngElement, true);
                    setSummaryPanelControlVisible(this.$scope, "ApprovalLinkDivAutoField",
this.$scope.ngElement, false);
                    $("#ApprovalLinkAutoField1", this.$element.parent().css("margin-top", "8px");
                } else {
                    setSummaryPanelControlVisible(this.$scope, "ApprovalLinkAutoField",
this.$scope.ngElement, false);
                    setSummaryPanelControlVisible(this.$scope, "ApprovalLinkDivAutoField",
this.$scope.ngElement, true);
                    $("#ApprovalLinkAutoField1", this.$element).parent().css("margin-top", "0px");
                }
            }

        }
```

Changing the label on a field on the Summary Panel

setSummaryPanelLabel: sets the label of the passed in field name.

```
// first parameter is the id of the field that is in the summary panel
// second parameter is the new label that it should be changed to
// third parameter is the scopes element
setSummaryPanelLabel("Field2AutoField", "new label", this.$element);
```

Changing the value of a Summary Panel field

Updates value in Summary Panel

TypeScript
param {string} name - Name of field in Summary Panel
param {*} value - Value which should be displayed in Summary Panel
param {string} type - Type of value
param {string} format - Format of value("Yes/No" for boolean type, kendo format for decimal, integer, date and datetime)

```
this.ViewController.setSummaryControlValue("Field1AutoField", 10.3, "decimal", "####,###,###,##0.
00");
this.ViewController.setSummaryControlValue("Field1AutoField", true, "boolean", "Yes/No");
this.ViewController.setSummaryControlValue("Field1AutoField", "2019-08-07T00:00:00.000Z", "date",
"yyyy-MM-ddTHH:mm:ss.fffZ");
```

Main View

Getting access to the Main View

The main view itself has no real UI elements to access. All elements are accessed through the other wrapper objects such as the Toolbar, View grids, View fields, etc.

Most of the events are data handling event.

Events

Main view events are handled in the Main view UI event handler. The following functions can be implemented to handle form field events:

Method	Event Data Properties	Description
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 autosave (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 cancels 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	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

	handler.	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 metadata is clicked.
onActionChange (eventData: QraView.ActionChangeEventData)	**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 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.BeforeViewInitEventData)		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 its 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	Called prior to sending data to the server.

	subscriber has taken control of the event processing	
onGetKeyData(eventData: QraView.GetKeyDataEventData)	**eventData.keyValuePairs**: key /value pair object with key data	Called when getKeyData is called on the Qra view controller.
onAttachmentsDataBound(eventData: QraView.AttachmentsDataBoundEventData)	**TODO: complete** **eventData.keyset**: **eventData.action**: **eventData.complete**:	Called when the attachments panel's data is bound.
onBeforeAttachmentDataBound(eventData: QraView.BeforeAttachmentsDataBoundEventData)	**eventData.dataRow**: binding data	Called prior to binding the attachments panel's data.
onActivityFeedDataBound(eventData: QraView.ActivityFeedDataBoundEventData<TNgData>)	**eventData.data**: view data	Called when the view's data is bound, and the activity feed panel can start loading.

Properties/Functions

- Customizing breadcrumb labels
- Displaying a message above the form
- Customizing flash messages
- Customizing the screen when the view changes between Create and Maint

Customizing breadcrumb labels

The following functions can help with customizing the current and previous breadcrumb labels on a pop-up screen, and typically should be invoked in the pop-up screen's afterViewInit event.

Note: To control the initial setting of a pop-up screen's breadcrumb to be the same style as for Web UI Detail screens (i.e. showing key field for each level, and auto-collapsing the breadcrumbs when the number of pop-up levels becomes large), you can use the utility function createBreadcrumbHtml() added in FD16. For more information and example code, see Creating modal popups.

Get the label for the current (last) breadcrumb level.

```
getCurrentBreadcrumbLabel();
```

Get the label for the previous (second-to-last) breadcrumb level.

```
getPreviousBreadcrumbLabel();
```

Set the label for the current (last) breadcrumb level. Note: the input label string will be automatically HTML-character-escaped for security, so no pre-escaping should be done by the caller.

```
setCurrentBreadcrumbLabel($scope.QraLabelService.getLabel("COMPONENTS"));
```

Set the label for the previous (second-to-last) breadcrumb level. Parameters are the label to be shown and a boolean representing whether or not to include the key fields in the breadcrumb. Note: the input label string will be automatically HTML-character-escaped for security, so no pre-escaping should be done by the caller.

```
setPreviousBreadcrumbLabel($scope.QraLabelService.getLabel("PARENT"), true);
```

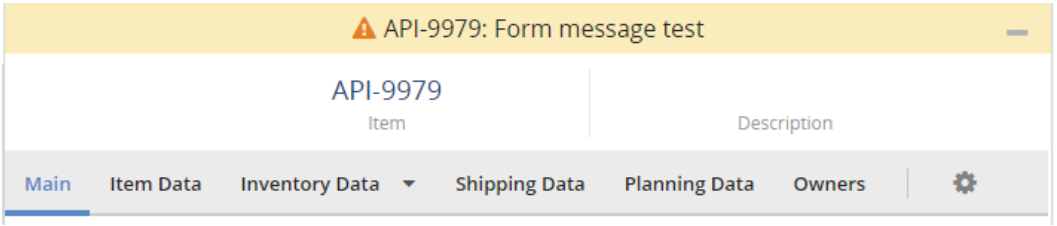
Get the full breadcrumb HTML for the current Kendo window level.

```
getKendoWindowCtrlr(getKendoWindowChildLevel()).iframeTitle
```

Example of using the above functions:

```
onAfterViewInit(eventData: EventData.QraView.AfterViewInitEventData): void {
    if (getKendoWindowChildLevel() > 0) {
        setCurrentBreadcrumbLabel(this.ViewController.getLabel("COMPONENTS"));
        let prevBreadcrumbLabel = getPreviousBreadcrumbLabel();
        if (prevBreadcrumbLabel.indexOf(this.ViewController.getLabel("COMPONENTS"))>=0)
            setPreviousBreadcrumbLabel("", true);
    }
}
```

Displaying a message above the form



Form messages are displayed above the form as in the picture above. To display one, displayFormMessage can be used in the ts-handler.

```
public onAfterBindData(eventData: Qad.Common.Service.Communication.EventData.QraView.
AfterBindDataEventData<any>): void {
        if (this.NgData.items[0].itemCode == "API-9979")
                this.ViewController.displayFormMessage("API-9979: Form message test");
        else
                this.ViewController.displayFormMessage(null);
}
```

Customizing flash messages

setSuccessFlashMessage: Sets the success flash message for next call

setErrorFlashMessage: Sets the error flash message for next call

The success AND error flash messages stay until the next flash message is displayed on the UI. This means the error flash message is reset to the UI default even after a successful call, and same for success on error. Since the updated messages stay until a flash message is displayed, only set the updated messages right before the call. This can be done in the BeforeDelete or the BeforeUpdate event such as shown below, or it can be done right before executing a save in handler code. If there are confirmations before the save/delete/update, make sure to set the updated message only after confirm.

Example:

```
public onBeforeUpdate(eventData: Qad.Common.Service.Communication.EventData.QraView.
BeforeUpdateEventData){
        if(this.NgData && this.NgData.commodityCodeMasters && this.NgData.commodityCodeMasters[0]
&&
        this.NgData.commodityCodeMasters[0].commodityCode == "API-12054" && this.NgData.
commodityCodeMasters[0].description != null) {
                this.ViewController.setSuccessFlashMessage(this.ViewController.getLabel("mfg-
SUCCESS"));
                this.ViewController.setErrorFlashMessage(this.ViewController.getLabel("mfg-
ERROR"));
        }
}
```

Customizing the screen when the view changes between Create and Maint

Setting a field Disabled:

```
if(eventData.newAction == "CREATE")
    this.ViewController.getViewField("SalesOrderNumberAutoField").IsDisabled=true;
else if(eventData.newAction == "UPDATE")
    this.ViewController.getViewField("SalesOrderNumberAutoField").IsDisabled=false;
```

Changing the label of a grid column:

```
onActionChange(eventData: EventData.QraView.ActionChangeEventData): void {
    if(eventData.newAction == "CREATE")
    {
        var columnLabel = this.ViewController.getLabel("LABEL_CREATE","qad-acme");
        columnLabel = escapeHtml(columnLabel);
        this.ViewController.getViewGrid("MyGrid").setColumnTitle("method",this.ViewController.
getLabel("LABEL_CREATE","qad-acme"));
    }
    else if(eventData.newAction == "UPDATE")
    {
        var columnLabel = this.ViewController.getLabel("LABEL_UPDATE","qad-acme");
        columnLabel = escapeHtml(columnLabel);
        this.ViewController.getViewGrid("MyGrid").setColumnTitle("method",this.ViewController.
getLabel("LABEL_CREATE","qad-acme"));
    }
}
```

Toolbar

Getting access to the toolbar

New way

```
const topToolbar = this.ViewController.TopToolbar;
const bottomToolbar = this.ViewController.BottomToolbar;
```

Old way

The wrapper object for the error viewer is accessible via the ViewController:

```
let toolbar = this.ViewController.ToolBar;
```

Events

Main UI event handler:

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,	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.

	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 autosave (true).	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.
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.ToolbarInitializedEventData)	null	Called when the toolbar is initialized.

Form event handler:

Method	Event Data Properties	Description
onButtonClick (viewButton: IViewButton, eventData: Button.ButtonClickEventData)	null	Called when a button defined in the view metadata is clicked or when a button defined using addBottomButton() is clicked.

Properties/Functions

Add a button to the bottom toolbar

New way

```
const options = {isSubmitButton: true, class: 'btn-color-primary'}; // Additional configuration
of method. You can add the text to class attribute and override button, which is clicked on form
submit.
this.ViewController.BottomToolbar.addButton('My bottom button1', 'MyBottomButton1');
this.ViewController.BottomToolbar.addButton('My bottom button2', 'MyBottomButton2', options);
```

The click event is then handled in the form event handler:

```
onButtonClick(viewButton: Qad.QraView.TSHandler.IViewButton, eventData: EventData.QraView.
ButtonClickEventData): void {
    if (eventData.buttonId=="MyBottomButton1") {
        alert("MyBottomButton1 is clicked !");
    }
}
```

Old way

The following code shows how to add and enable a button. Note that this is done in the onToolbarInitialized event handler in the main view controller:

```
onToolbarInitialized(eventData: EventData.QraView.ToolbarInitializedEventData): void {
    this.ViewController.ToolBar.addBottomButton("MyButton","My Button",false);
    this.ViewController.ToolBar.setToolbarItemDisabled("MyButton",false);
}
```

The click event is then handled in the form event handler:

```
onButtonClick(viewButton: Qad.QraView.TSHandler.IViewButton, eventData: EventData.QraView.
ButtonClickEventData): void {
    if (eventData.buttonId=="MyButton") {
        alert("MyButton is clicked !");
    }
}
```

Add option buttons to a bottom toolbar button

New way

```
this.ViewController.BottomToolbar.addButton('My bottom button1', 'MyBottomButton1');
this.ViewController.BottomToolbar.addOptionButton('My option', 'OptionButtonId',
'MyBottomButton1');
```

The click event is then handled in the form event handler:

```
onButtonClick(viewButton: Qad.QraView.TSHandler.IViewButton, eventData: EventData.QraView.
ButtonClickEventData): void {
    if (eventData.buttonId=='MyBottomButton1') {
        alert("MyBottomButton1 is clicked !");
    }
}
```

Old way

The following code shows how to add option buttons to a button. Note that this is done in the onToolbarInitialized event handler in the main view controller:

```
onToolbarInitialized(eventData: EventData.QraView.ToolbarInitializedEventData): void {
    this.ViewController.ToolBar.addBottomButton("MyButton","My Button",false);
    this.ViewController.ToolBar.setToolbarItemDisabled("MyButton",false);
    this.ViewController.ToolBar.addOptionButton("MyButton","Action 1","Action1");
    this.ViewController.ToolBar.addOptionButton("MyButton","Action 2","Action2");
}
```

The click event is then handled in the form event handler:

```
onButtonClick(viewButton: Qad.QraView.TSHandler.IViewButton, eventData: EventData.QraView.
ButtonClickEventData): void {
    if (eventData.buttonId=="MyButton") {
        alert("MyButton is clicked !");
    }
    else if (eventData.buttonId=="MyButtonOption1") {
        alert("MyButton option 1 is clicked !");
    }
    else if (eventData.buttonId=="MyButtonOption2") {
        alert("MyButton option 2 is clicked !");
    }
}
```

Show/Hide the button of a bottom toolbar

```
const buttonId = 'SomeId';
const button = this.ViewController.BottomToolbar.getToolbarItem(buttonId);
//Hide
button.IsVisible = false;
//Show
button.IsVisible = true;
```

Show/Hide the option button of a bottom toolbar

```
const parentButtonId = 'SomeId';
const optionButtonId = 'someChildOptionButtonId'
const optionButton = this.ViewController.BottomToolbar.getToolbarItem(optionButtonId,
parentButtonId);
//Hide
optionButton.IsVisible = false;
//Show
optionButton.IsVisible = true;
```

Enable/Disable the button of a bottom toolbar (It does not work for the default Submit and Cancel buttons)

```
const buttonId = 'SomeId';
const button = this.ViewController.BottomToolbar.getToolbarItem(buttonId);
//Disable
button.IsEnabled = false;
//Enable
button.IsEnabled = true;
```

Enable/Disable the option button of a bottom toolbar

```
const parentButtonId = 'SomeId';
const optionButtonId = 'someChildOptionButtonId'
const optionButton = this.ViewController.BottomToolbar.getToolbarItem(optionButtonId,
parentButtonId);
//Disable
optionButton.IsEnabled = false;
//Enable
optionButton.IsEnabled = true;
```

Change the text of the button of a bottom toolbar

```
const buttonId = 'SomeId';
const button = this.ViewController.BottomToolbar.getToolbarItem(buttonId);
//Read text
console.log(button.Text);
//Change text
button.Text = 'New'text;
```

Change the text of option button of a bottom toolbar

```

const parentButtonId = 'SomeId';
const optionButtonId = 'someChildOptionButtonId'
const optionButton = this.ViewController.BottomToolbar.getToolbarItem(optionButtonId,
parentButtonId);

//Read text
console.log(optionButton.Text);
//Change text
optionButton.Text = 'New'text;
```

Add a button to the top toolbar

New way

```

```

```
this.ViewController.TopToolbar.addButton('Some button', 'SomeButtonId');
const options = {icon: 'fa fa-plus', onClick: event => console.log(event), class: 'my'};
this.ViewController.TopToolbar.addButton('Some button1', 'SomeButtonId1', options);
```

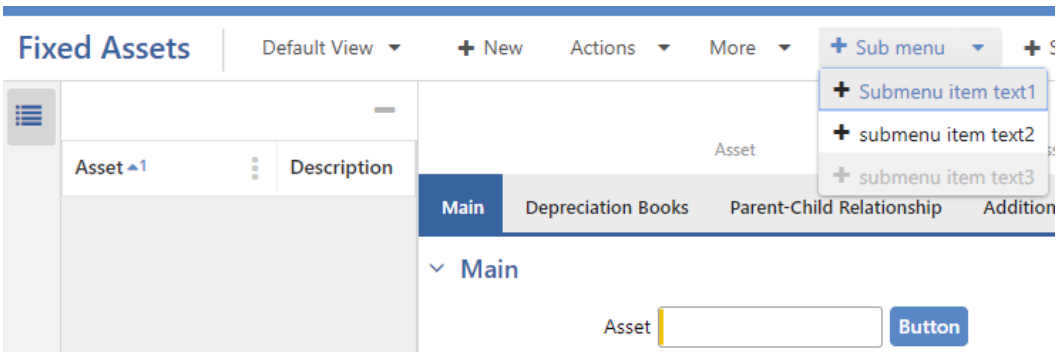
Old way

Note that you have to attach a click event handler. Tool buttons added in this way don't fire the framework click event.

```
onBeforeToolbarInitialized(eventData: EventData.QraView.BeforeToolbarInitializedEventData): void {
    let item=Qad.QraView.UI.ToolBar.createToolbarItem("MyToolbarItem","My toolbar item");
    item.bottomBar=false;
    eventData.toolbarData.toolbarItems.push(item);
}

onToolbarInitialized(eventData: EventData.QraView.ToolbarInitializedEventData): void {
    this.addEventHandler($("#MyToolbarItem",,this.$element.closest("#qMainContainer")),"click",(e)
=> {
        alert("MyToolbarItem is clicked !");
    });
}
```

Add a sub-menu to the top toolbar

```
const title = 'Sub menu;
const id = 'SubMenuId';
const submenuItems = [
    {id: 'Submenuitem1', text: 'Submenu item text1', icon: 'fa fa-plus', class: 'my', onClick:
event => console.log(event, 1)},
    {id: 'Submenuitem2', text: 'submenu item text2', icon: 'fa fa-plus', onClick: event =>
console.log(event, 2), isEnabled: true, isVisible: true},
    {id: 'Submenuitem3', text: 'submenu item text3', icon: 'fa fa-plus', isEnabled: false,
isVisible: true},
];
const options = {icon: 'fa fa-plus', class: 'my'};

this.ViewController.TopToolbar.addSubMenu(title, id, submenuItems, options);
```



Show/Hide the button or the sub-menu of a top toolbar

```
const buttonId = 'SomeId';
const button = this.ViewController.TopToolbar.getToolbarItem(buttonId);
//Hide
button.IsVisible = false;
//Show
button.IsVisible = true;
```

Show/Hide the sub-menu item of a top toolbar

```
const submenuId = 'SomeId';
const submenItemId = 'someChildOptionButtonId'
```

```
const submenItem = this.ViewController.TopToolbar.getToolbarItem(submenItemId, submenuId);
//Hide
submenItem.IsVisible = false;
//Show
submenItem.IsVisible = true;
```

Enable/Disable the button or the sub-menu of a top toolbar

```
const buttonId = 'SomeId';
const button = this.ViewController.TopToolbar.getToolbarItem(buttonId);
//Disable
button.IsEnabled = false;
//Enable
button.IsEnabled = true;
```

Enable/Disable the sub-menu item of a top toolbar

```
const submenuId = 'SomeId';
const submenItemId = 'someChildOptionButtonId'
const submenItem = this.ViewController.TopToolbar.getToolbarItem(submenItemId, submenuId);
//Disable
submenItem.IsEnabled = false;
//Enable
submenItem.IsEnabled = true;
```

Change the text of the button or the sub-menu of a top toolbar

```
const buttonId = 'SomeId';
const button = this.ViewController.TopToolbar.getToolbarItem(buttonId);
//Read
console.log(button.Text);
//Write
button.Text = 'New text';
```

Change text of the sub-menu item of a top toolbar

```
const submenuId = 'SomeId';
const submenItemId = 'someChildOptionButtonId'
const submenItem = this.ViewController.TopToolbar.getToolbarItem(submenItemId, submenuId);
//Read
console.log(submenItem.Text);
//Write
submenItem.Text = 'New text';
```

Hybrid View

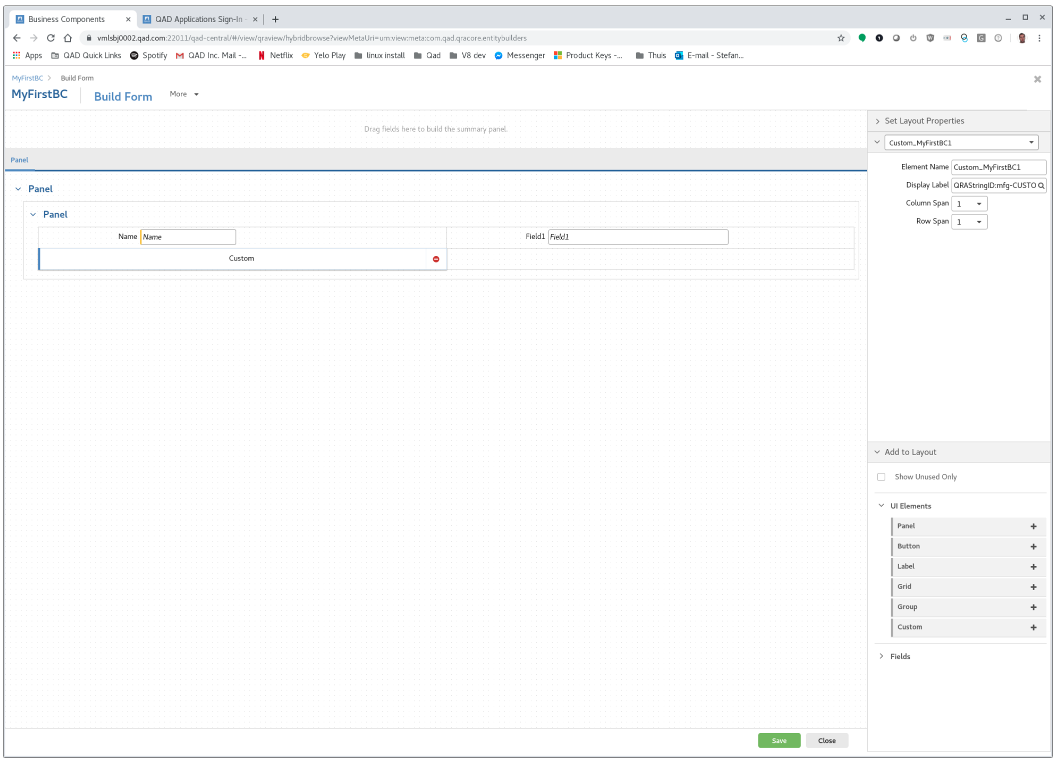
Hybrid view events:

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.

Custom control

What is a custom control

A custom control can be added to a view in the form builder:



It can be used as a placeholder to add custom html.

Getting access to a Custom control object

A custom control needs to be accessed with JQuery. The HTML at runtime looks as follows:

```
<td style="height:27px; width:50%;"><div class="qraViewFormFld" id="Custom_MyFirstBC1" qvisible="
1"><label class="qraViewFormLabel" hidden="">Custom</label></div></td>
```

So, in order to get the element, you can run this JQuery:

```
let customElement=$("#Custom_MyFirstBC1",this.$element);
```

Adding custom HTML to a custom control

A custom control can be used to add custom HTML:

```
        protected init(): void {
            $("#Custom_MyFirstBC1",this.$element).append('<img class="logo" src="resources/qad
/webshell/style/img/QAD_Logo.png">');
        }
```

Common functions

These pages describe some functions that are more general, and can be used in every UI event handler:

- Making server calls
- Showing messages
- Showing a confirmation dialog
- Getting the current User
- Utility functions

Making server calls

Introduction

We often do http calls to server api's to get or save data. It is important that this happens in a controlled matter, and always goes through the same logic. This page explains how to do this.

Calls done this way are handled by the framework and have the following advantage:

- When closing a pop-up screen, the framework waits five seconds for the calls to end. If not finished after five seconds, they are canceled and the pop-up screen closes.
- The calls can be canceled easily.
- The calls can use standard error handling or an error callback.

doHttp methods

There are three doHTTP methods that can be used on view controllers and view classes. All the methods return an HttpCallController object that can be used to cancel the call and read some status information.

The methods have the following common parameters:

1. successCallback: runs when the call finished successfully.
2. errorCallback: runs when the call returned with an error.
3. config: ng.IRequestShortcutConfig, can be used to configure the call (see https://docs.angularjs.org/api/ng/service/$http#usage), mainly used to add headers.
4. skipDefaultErrorHandling: When set to false, a failed call will show the standard error page (full screen error). When set to true, no error page is shown and the error can be handled in the error callback. Default is false.

The return value of the calls is a HttpCallController object that can be used to cancel the call or get some info about the call:

```
        public get Config(): ng.IRequestShortcutConfig {

        public get IsFinished(): boolean {

        public get IsCancelled(): boolean {

        public get IsFinishedSuccessFully(): boolean {

        public get UID(): string {

        public cancel(reason: string): void {
```

example:

```
        public testTest(appURI: string) {

            this.ViewController.doHttpGet(
                bindTemplateData("api/qracore/apps?appURI={appURI}", { appURI: appURI }),
                (responseData: MyTypedData, status: number, headers, config) => {

                    cnost tableData : MyTypedDataRecord = responseData.table
[0];
                    const appName : string = tableData.appName;
                    ...
                }
            );
```

```
    }


    ...

    export interface APIResponse {
    submitResult: SubmitResult;
    success: boolean;
}

export interface MyTypedData extends APIResponse {
    table:MyTypedDataRecord[];
}
export interface MyTypedDataRecord {
    appURI: string;
    appName: string;
    description: string;
    stringCode: string;
    isDefault: boolean;
    appVersion: string;
    qadEnterpriseAppExportVersion: string;
    appKey: string;
    displayName: string;
    concurrencyHash: string;
    dataOperation: string;
    disallowedActions: string;
    disallowedActionsMessage: string;
}
```


The data in the response DataResult object is of type any. It should be cast to the correct type like in the above example.

The three methods are:

1. ## doHttpGet

   This method can be used to do an asynchronous http get call. Note the useCache parameter, if set to true, the browse cache result can be used.

```
         /**
         * 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.
         */
        public doHttpGet(url: string, successCallback: ng.IHttpPromiseCallback<{}>,
errorCallback: ng.IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController:
HttpCallController)=>void,config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?:
boolean,useCache?: boolean): HttpCallController {
```

2. ## doHttpPost

   This method can be used to do a http post call.

```
       /**
         * 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.
         */
        public doHttpPost(url: string, successCallback: ng.IHttpPromiseCallback<{}>,
errorCallback: ng.IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController:
HttpCallController)=>void,data?, config?: ng.IRequestShortcutConfig,
skipDefaultErrorHandling?: boolean): HttpCallController {
```

3. ## doHttpDelete

This method can be used to do a http delete call.

```
        /**
         * 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.
         */
        public doHttpDelete(url: string,successCallback: ng.IHttpPromiseCallback<{}>,
errorCallback: ng.IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController:
HttpCallController)=>void,config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?:
boolean): HttpCallController {
```

Showing messages

Introduction

There are a few different ways to show a message on the UI. We can show a dialog box, a flash message, and a message in the message log at the right of the screen. These three types of messages can be created with the same api call, which is explained on this page.

showMessage api function

On the base class of the UI event handlers, there is a method to show messages:

public showMessage(options: any, messages: any)

parameters:

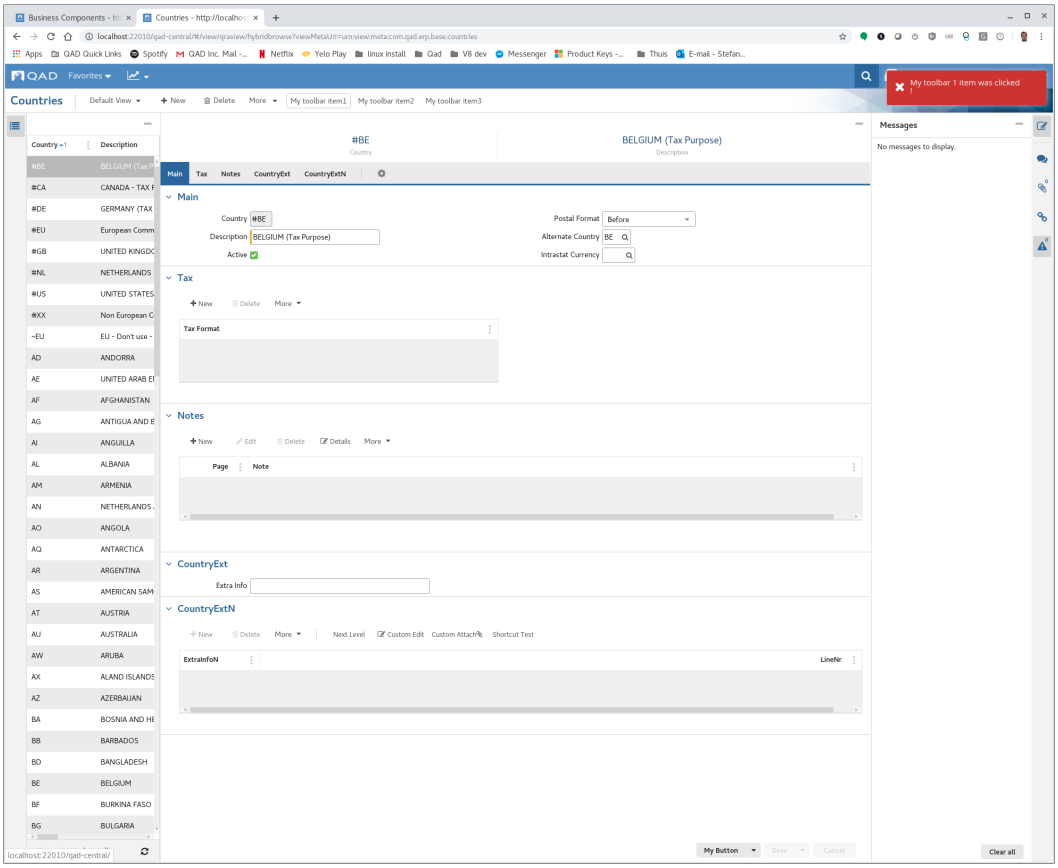
- options: JSON object of options:
  - action (optional): String action: Valid values:
    - "ADD" - Append message to message Log (Default Action if action not passed.)
    - "REMOVE" - Remove message from message Log. Note that messageId must be supplied in messages data.
    - "CLEAR" - Clear message Log.
  - showFlash (optional): boolean to show flash/toastr message (default = true)
  - showDialog (optional): boolean to show messages in dialog box (default = false). Note that if value is true, messages will not be shown in message Log.
  - addToMessageLog (optional): boolean to add the messages to the message log action. If set to true, the action (see first option) is applied to all the messages in messages parameter. If set to false, the messages are ignored for the action.
- messages: List of message JSON objects to display:
  - messageId (optional): message identifier
  - severity: number message severity
  - message: string message text

Examples

Show a flash message and do not log the message to the message log

code:

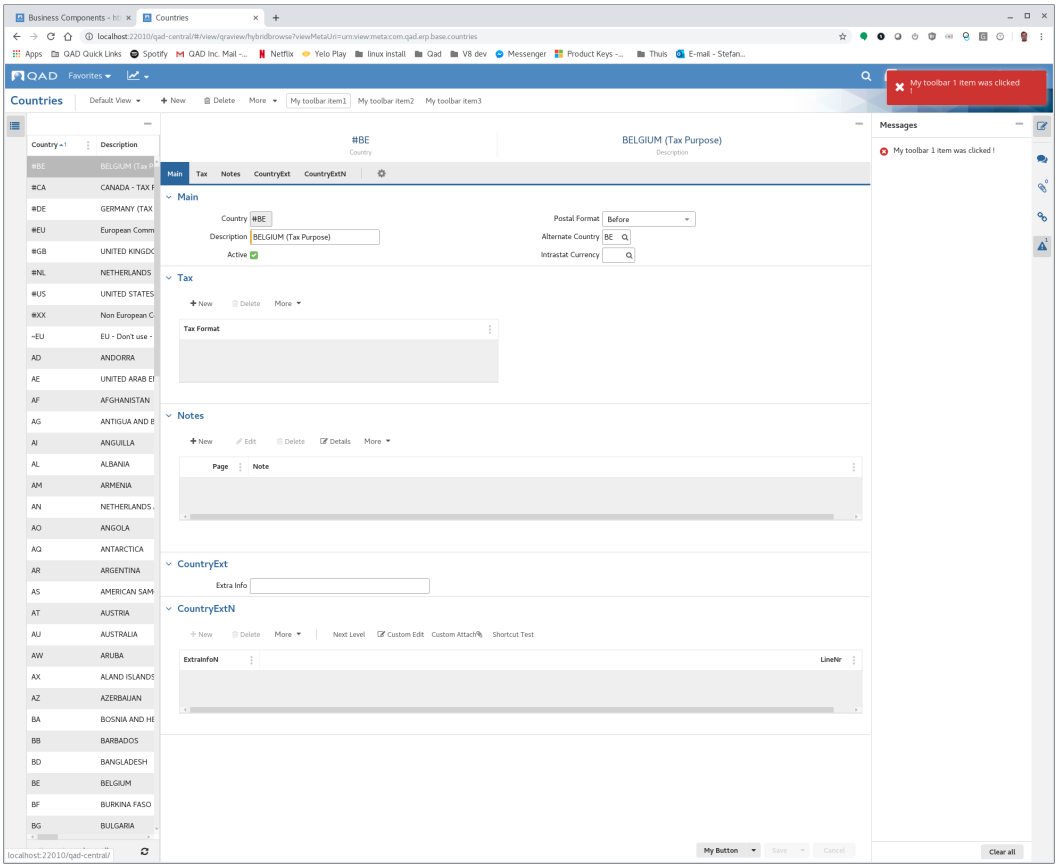
```
let options={action:"ADD",showFlash:true,showDialog:false,addToMessageLog:false};
let message={messageId:1,severity:1,message:"My toolbar item was clicked !"};
this.showMessage(options,[message]);
```



Show a flash message and log the message to the message log

code:

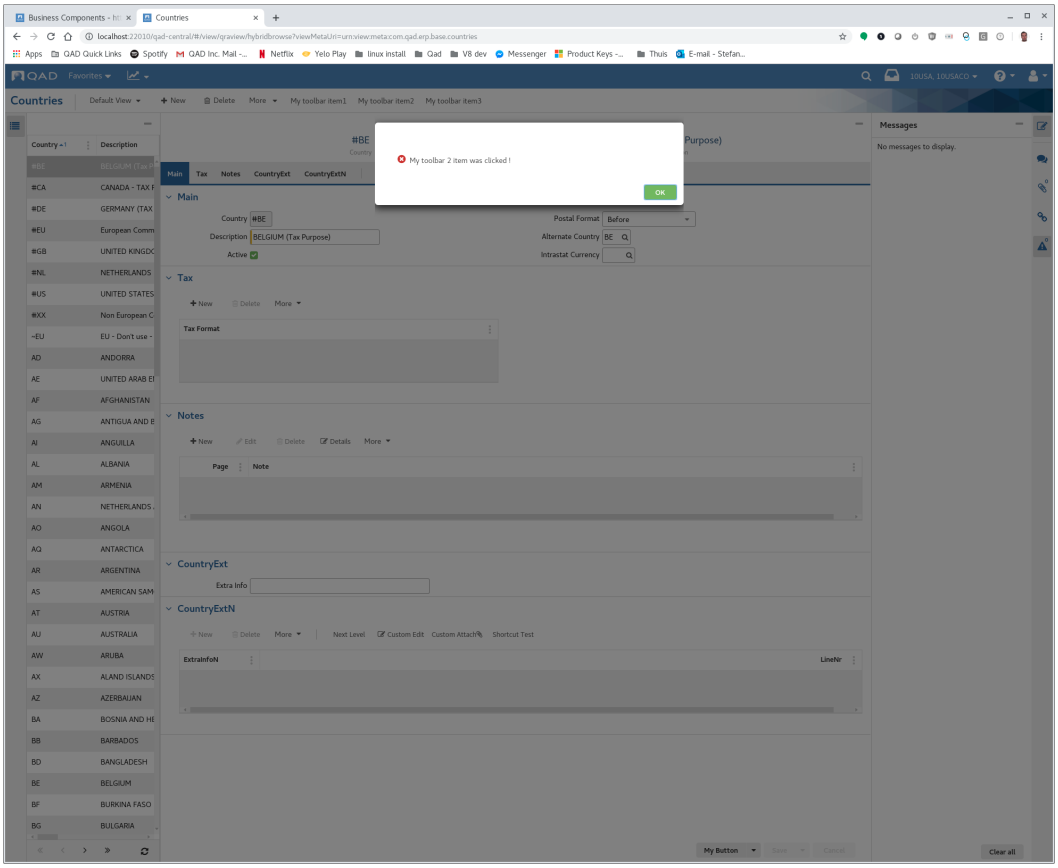
```
let options={action:"ADD",showFlash:true,showDialog:false,addToMessageLog:true};
let message={messageId:1,severity:1,message:"My toolbar item was clicked !"};
this.showMessage(options,[message]);
```



Show a dialog message

code:

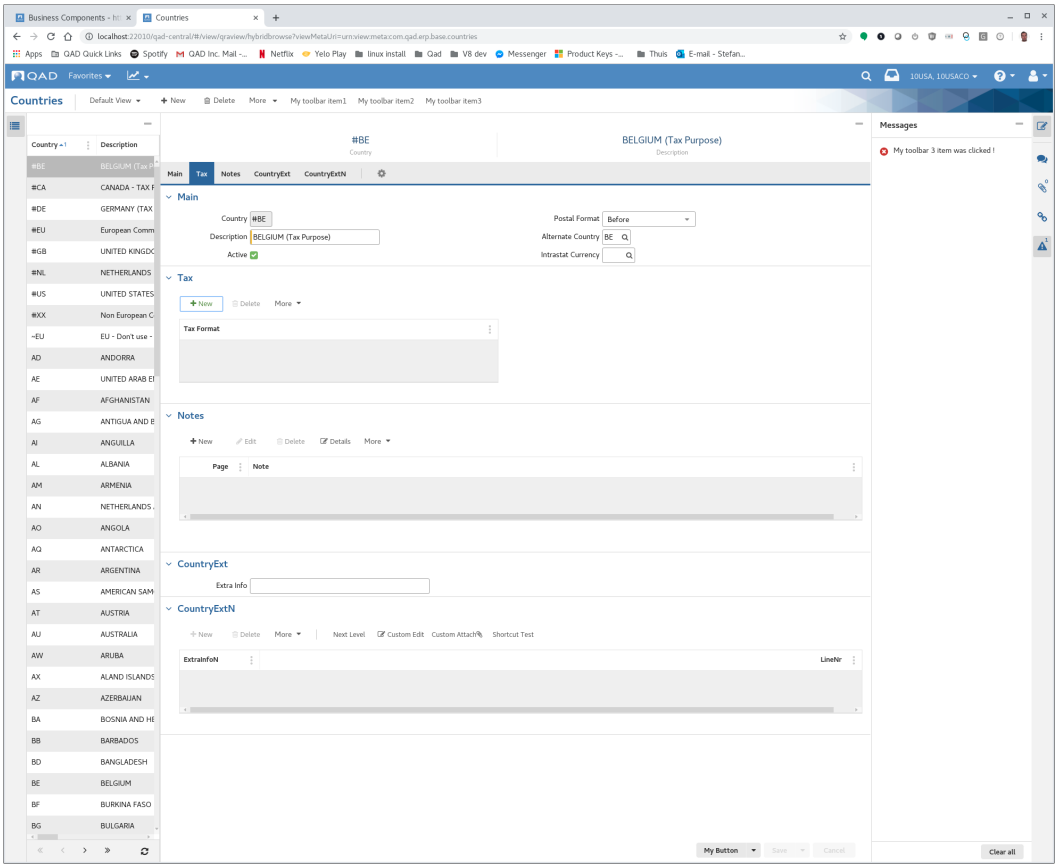
```
let options={action:"ADD",showFlash:false,showDialog:true,addToMessageLog:false};
let message={messageId:1,severity:1,message:"My toolbar 2 item was clicked !"};
this.showMessage(options,[message]);
```



Add a message to the message log but don't show a flash message

code:

```
let options={action:"ADD",showFlash:false,showDialog:false,addToMessageLog:true};
let message={messageId:1,severity:1,message:"My toolbar 3 item was clicked !"};
this.showMessage(options,[message]);
```



Clear the message log

code:

```
let options={action:"CLEAR",showFlash:false,showDialog:false,addToMessageLog:false};
this.showMessage(options,undefined);
```

Showing a confirmation dialog

Introduction

This page explains how to show a confirmation message.

launchQModalDialogV2 api function

On the base class of the UI event handlers, there is a method to show messages:

public launchQModalDialogV2(modalOptions: Qad.WebShell.UI.QModalDialogModalOptions, preventEscaping?: boolean)

parameters:

- modalOptions: JSON object of options:
  - messageType?: string; optional. Can be one of the following:
    - "Error"
    - "Info" (default)
    - "Question"
    - "Warning"
    - "Default"
  - confirmButtonText?: string; // MUST be the label term, NOT the translated value , defaults to mfg-OK
  - closeButtonText?: string; // MUST be the label term, NOT the translated value, defaults to mfg-CANCEL
  - showConfirmButton?: boolean; default=true
  - showCloseButton?: boolean; default=true
  - title?: string;
  - messageText: string;
  - confirmClickFunction?: ()=>void; : function that gets called when confirm is clicked
  - confirmFocusObj?: JQuery | string; : object that will get focus when confirm is clicked
  - closeClickFunction?: ()=>void; : function that gets called when close is clicked
  - closeFocusObj?: JQuery | string; : object that will get focus when close is clicked
- preventEscaping (optional): message identifier
  - severity: number message severity
  - message: string message text

example:

```
private showConfirmationDialog() {

    var title = "Dialog Title";
    var messageText = "Are you sure ?";

    // Disable form fields after edit cancel
    var closeClickFunction = ()=> {
                //do action on close ( cancel )
    }

    var confirmClickFunction = ()=> {
                //do action on confirm
    }


    // Prompt for edit confirmation
    this.launchQModalDialogV2({
        messageType: qModalDialogType.Question,
        title: title,
        messageText: messageText,
        confirmButtonText: "mfg-YES", // defaults to mfg-OK
        closeButtonText: "mfg-NO", // defaults to mfg-CANCEL
        showConfirmButton: true,
        showCloseButton: true,
        confirmClickFunction: confirmClickFunction,
        closeClickFunction: closeClickFunction
```

```
    });
}
```

example 2: call confirm and close methods on the class.

```
        private showConfirmationDialog() {

            var title = "Dialog Title";
            var messageText = "Are you sure ?";


            // Prompt for edit confirmation
            this.launchQModalDialogV2({
                messageType: qModalDialogType.Question,
                title: title,
                messageText: messageText,
                confirmButtonText: "mfg-YES", // defaults to mfg-OK
                closeButtonText: "mfg-NO", // defaults to mfg-CANCEL
                showConfirmButton: true,
                showCloseButton: true,
                confirmClickFunction: ()=> {this.onDialogConfirm()},
                closeClickFunction: ()=> {this.onDialogCancel()}
            });

        }

        private onDialogConfirm() {
            //do action on confirm
            let options={action:"ADD",showFlash:true,showDialog:false,addToMessageLog:true};
            let message={messageId:1,severity:1,message:"Confirm !"};
            this.showMessage(options,[message]);
        }

        private onDialogCancel() {
            //do action on close ( cancel )
            let options={action:"ADD",showFlash:true,showDialog:false,addToMessageLog:true};
            let message={messageId:1,severity:1,message:"Cancel !"};
            this.showMessage(options,[message]);
        }
```

Dialog box:

Getting the current User

Introduction

This page explains how to show a confirmation message.

Getting the current user

```
let currentUserId=this.$scope.getViewController().QraUserService.getUserId();
let currentUserFullName=this.$scope.getViewController().QraUserService.getUserFullName();
```

Utility functions

Introduction

This page describes some utility functions that are available for UI event handlers.

setFocusToField

Setting focus to a control. This function encapsulates the different handling required to set focus to different types of controls, including the various Kendo controls which require special focus handling.

```
this.ViewController.getViewField("MyField").focus();
//The second optional parameter is a boolean. If "true" then the contents of the field being
focused will be selected.
this.setFocusToField(this.ViewController.getViewField("MyField").Element),true);
```

resizeFieldWrapper

Resizing the border surrounding a field to exactly fit the field and the corresponding lookup button if one exists. This function should be called whenever a field's width or height is manually changed. The inputs are the field that has been resized, and a parameter called forceResize that denotes whether the resize should be forced to occur. Note: the framework will default forceResize to "false", so "true" must always be explicitly passed in.

```
resizeFieldWrapper(this.ViewController.getViewField("MyField").Element, true);
```

publishBrowseRefreshRequestEvent

Request that the browse, in a hybrid screen, be refreshed.

WARNING: This utility is to be used very infrequently and only in cases where without a doubt, the browse needs additional refreshes beyond those done by the framework. Browse refreshes are very expensive for performance.

```
//Request a browse refresh and re-position to the currently selected row.
publishBrowseRefreshRequestEvent(this.$scope, 0);

OR

//Request a browse refresh and re-position to the next row after the currently selected row
publishBrowseRefreshRequestEvent(this.$scope, 1);

//Note: the $scope in the two calls above has to be the "maint" scope.
//So if this is being called from the browse TShandler then use this.$scope.getMaintView() to get
```

```
the maint scope.

//Also, starting in FD19 there is also a 3rd optional parameter "publishRowChange" that defaults
to "false".
//If set to "true" then the browse will publish a rowChange event which will also cause the
maintenance screen to refresh. For example:
publishBrowseRefreshRequestEvent(this.$scope, 0, true);

//In FD35 the browse can be re-positioned to a specific row based on key values. In this example
"commodityCodeMaster.CommodityCode"
//is the browse column name and "001" is the Commodity Code key field value.
publishBrowseRefreshRequestEvent(this.$scope, RowSelection.ByKeys, true, {"commodityCodeMaster.
CommodityCode": "001"});
```

publishHybridConfigureEvent

Override the default browse refresh behavior when saving a record.

WARNING: This utility is to be used only in cases where without a doubt, the default browse refresh pattern should be overridden. Browse refreshes are very expensive for performance.

```
public onAfterViewInit(eventData: Qad.Common.Service.Communication.EventData.QraView.
AfterViewInitEventData){
    //The default behavior of the hybrid controller is to not refresh the browse
    //after saving an existing record. This call overrides that hybrid controller
    //setting so that it refreshes the browse after saving an existing record.
    publishHybridConfigureEvent(this.$scope, true);
}
```

executeSaveAction

Save the form. This is equivalent to pressing the "Save" button at the bottom of the form. Use this function instead of finding the "Save" button on the screen and triggering a click on it. Triggering the click can have unintended side-effects like causing the "Save & Next" to execute instead of just the "Save".

There is an optional parameter "simulateScreenClick" (boolean) with a default value of *false*. Passing *true* to "simulateScreenClick" will cause browse to refresh.
There is a second optional parameter "ignoreActivePopup" (boolean) with a default value of *false.* The Framework will not allow the save to occur if a pop-up screen is active. So this parameter, when set to *true*, will override this behavior and allow the save to occur.

```
this.ViewController.executeSaveAction()
```

show/hide loading image

Show/Hide loading image (spinning icon), over the current UI

```
this.ViewController.showLoadingImage();
this.ViewController.hideLoadingImage();
```

Show/Hide loading image (spinning icon), over the entire browser screen

showLoadingImageFullCoverage(onlyDisplayInDialog?: any): void

onlyDisplayInDialog: optional, true if only the current dialog box should have a waiting cursor, default is false.

example:

```
this.ViewController.showLoadingImageFullCoverage();
```

isValidEmail

isValidEmail: determines if the given email address is valid or not.

```
/**
 * isValidEmail: Determines if the given email address is valid or not.
 *
 * @param email: email address string
 * @returns true or false
 */

var isValid = isValidEmail("azr@qad.com");
```

getHybridBrowseViewUrl

Returns the correct HybridBrowse Url considering the device. For example, for IPad and mobile, the URL desired is in Maint HybridMode, whereas in Desktop, URL is desired in Hybrid mode.

getHybridBrowseViewUrl

```
/**
 * Construct the Hybrid view url for the entity from the entity view metadata URI.
 *
 * @param viewMetaUri the view metadata uri
 * @param keys the primary keys for the entity
 * @param params other parameters
 *
 * @return the view url
 */
var keys = {domainCode : this.NgData.approvals[0].domainCode, inventoryStatusCode : this.NgData.
approvals[0].approvalObject.inventoryStatusCode};
var url = getHybridBrowseViewUrl("urn:view:meta:com.qad.acme.inventoryStatusCodes",keys);
```

Session info

Getting access to the Session info

How to access the session info object

```
const sessionInfo = this.ViewController.SessionInfo;
```

Properties/Functions

Get the current user full name

Get the current user full name

```
console.log(sessionInfo.CurrentUserFullName); //Brad Pitt
```

Get the current user ID

Get the current user ID

```
const currUserId = sessionInfo.CurrentUserID
```

Get the current user role IDs

Get the current user roles ids

```
sessionInfo.CurrentUserRolesIDs // ['roles1Id', 'roles2Id', ...]
```

Display message manager (DisplayMessageManager)

Getting access to the DisplayMessageManager

Getting access to the DisplayManager

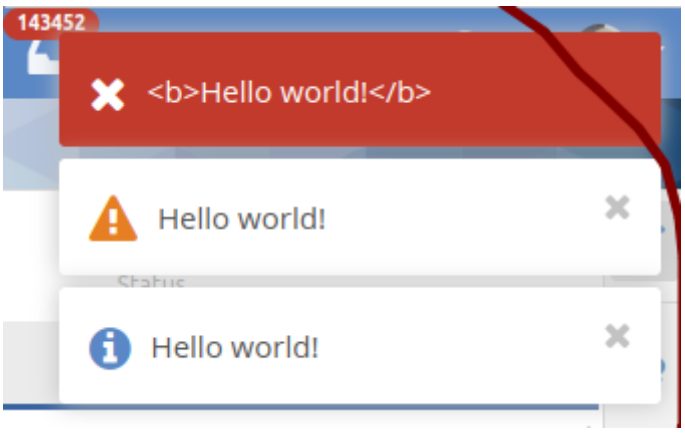
```
const displayManager = this.ViewController.DisplayMessageManager;
```

Properties/Functions

Show the flash messages

Show the flash message

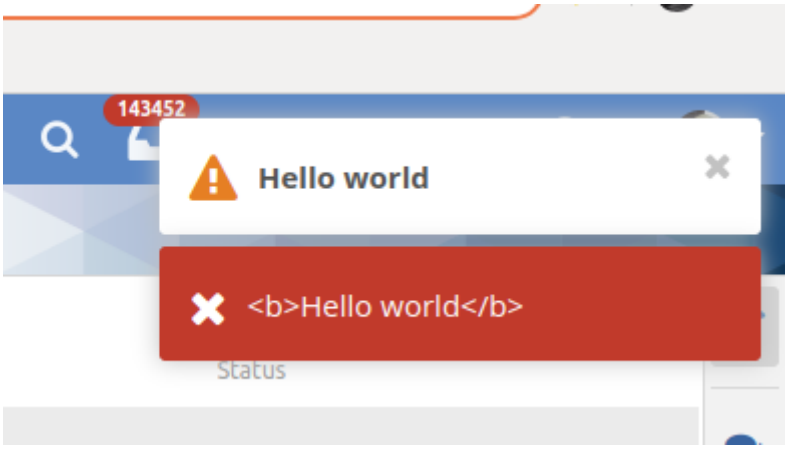
```
const text = 'Hello world';
const type = 'info';
displayManager.showFlashMessage(text, type);
displayManager.showFlashMessage('Hello world', 'warning');
displayManager.showFlashMessage('<b>Hello world</b>', 'error');
```



Show the flash messages with the doNotEscape param

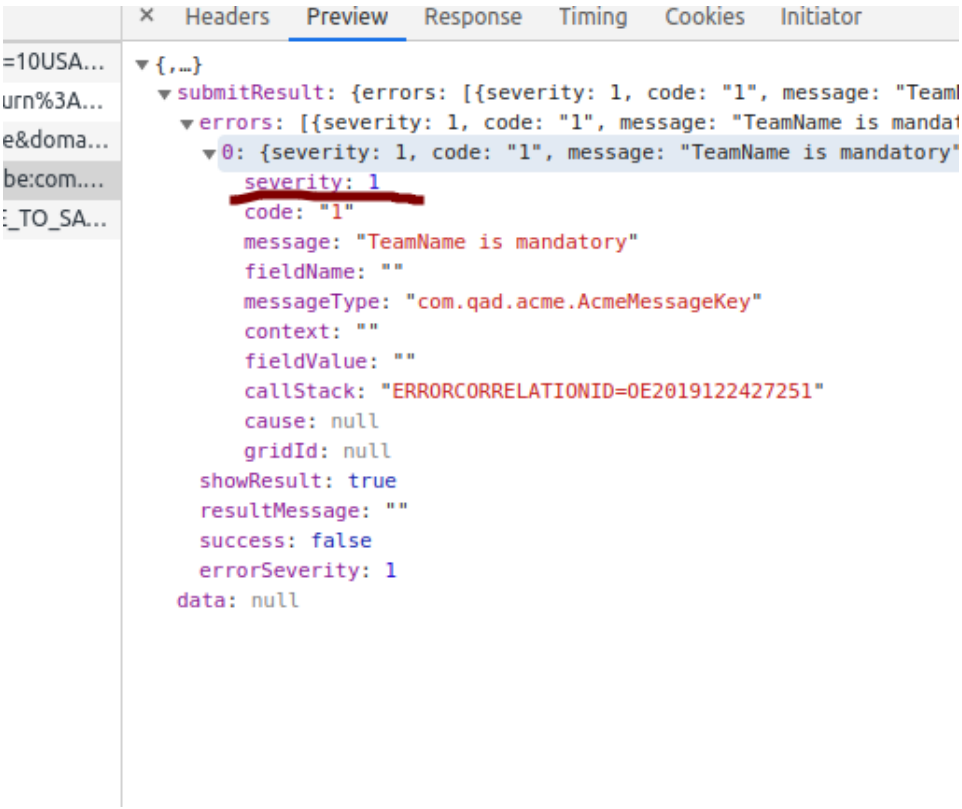
Show the flash messages with doNotEscape param

```
let doNotEscape = false;
displayManager.showFlashMessage('<b>Hello world</b>', 'error', doNotEscape);
doNotEscape = true;
displayManager.showFlashMessage('<b>Hello world</b>', 'warning', doNotEscape);
```



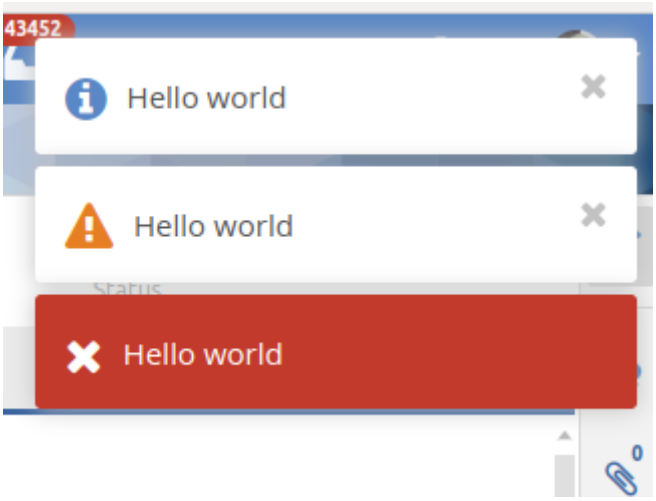
Show the flash message with type number (severity from the error)

We can use a number from the server response and put it in our function as a parameter **type**.



Show the flash message with type number (severity from the error)

```
displayManager.showFlashMessage('Hello world', 1);
displayManager.showFlashMessage('Hello world', 2);
displayManager.showFlashMessage('Hello world', 3);
```



Get the last flash message text

Get the last flash message text

```
displayManager.showFlashMessage('Message1', 1);
displayManager.showFlashMessage('Message2', 2);
displayManager.showFlashMessage('Message3', 3);

console.log(displayManager.getLastFlashMessageText()); //'Message3'
```

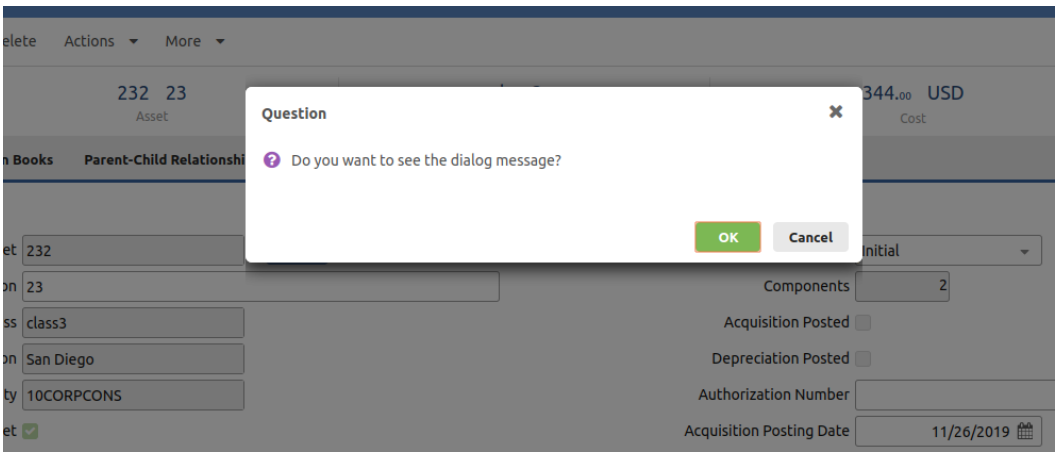
Show the dialog message

text - the same parameter like in Show the flash message

type - the same parameter like in Show the flash message except there is one new type **'question'**

Show dialog message

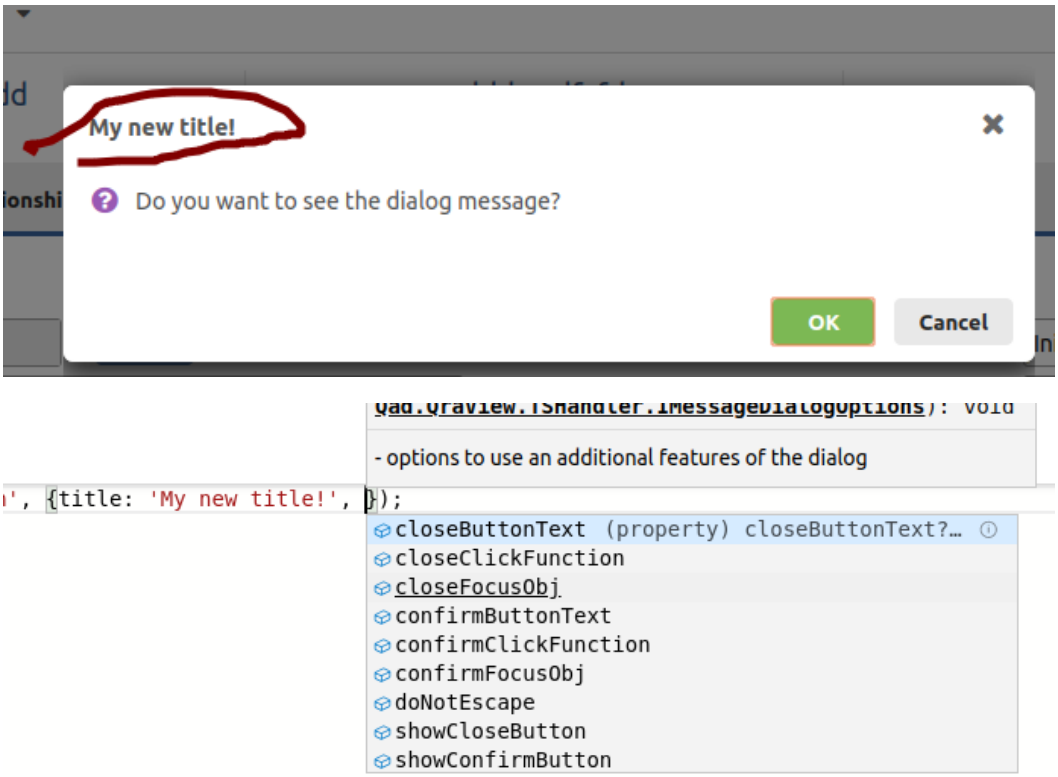
```
displayManager.showDialogMessage('Do you want to see the dialog?', 'question');
```



You can also use options when showing the dialog message.

Show the dialog message with options parameter

```
displayManager.showDialogMessage('Do you want to see the dialog message?', 'question', {title:
'My new title!'});
```



Manipulate the message log

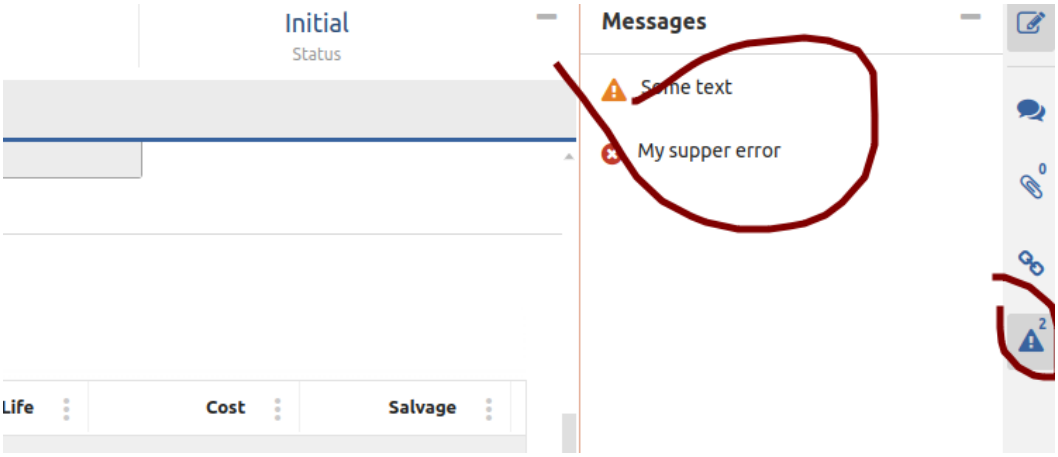
addMessage

text - the same parameter like in Show the flash message

type - the same parameter like in Show the flash message

Manipulate the message log

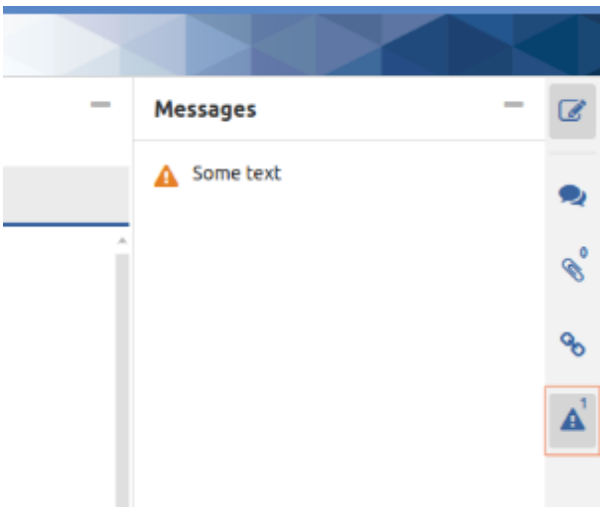
```
const messageId = 'id1';
const text = 'Some text';
const someType = 'warning'; // someType = 2; - the same
displayManager.MessageLog.addMessage(text, someType);
displayManager.MessageLog.addMessage('My supper error', 'error', messageId);
```



removeMessage (remove the message by ID)

Remove message from log

```
displayManager.MessageLog.removeMessage(messageId);
```



clear (delete all messages)

Clear all messages

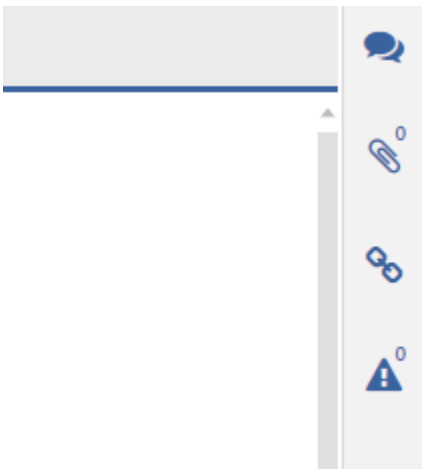
```
displayManager.MessageLog.clear();
```

Side panel

Getting access to the side panel

Getting access to the SidePanel

```
const sidePanel = this.ViewController.SidePanel;
```



Show/Hide the side panel icons

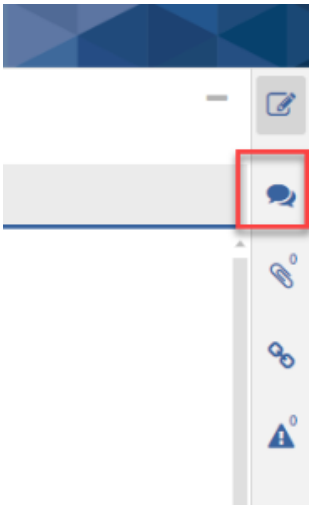
Show/hide a side panel

```
sidePanel.IsVisible = true; //Shows all elements of a side panel and change a property
"IsVisible" of these sub-elements to true: attachments, drilldowns, messages, and activity.
sidePanel.IsVisible = false; //Hides all elements of a side panel and change a property
"IsVisible" of these sub-elements to false: attachments, drilldowns, messages, and activity.
console.log(sidePanel.IsVisible); //If any element (attachments, drilldowns, messages, and
activity) is visible, return true.
```

Getting access to the activity feed panel

Getting access to the ActivityFeedPanel

```
const activityFeedPanel = this.ViewController.SidePanel.ActivityFeedPanel;
```



Show/Hide the activity feed panel icon

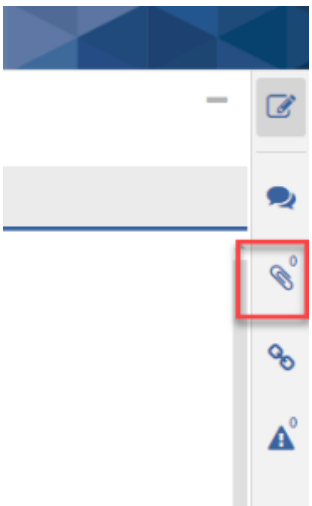
Show/hide the activity feed panel

```
activityFeedPanel.IsVisible = true;
activityFeedPanel.IsVisible = false;
```

Getting access to the attachments panel

Getting access to the attachments panel

```
const attachmentsPanel = this.ViewController.SidePanel.AttachmentsPanel;
```



Show/Hide the attachments panel icon

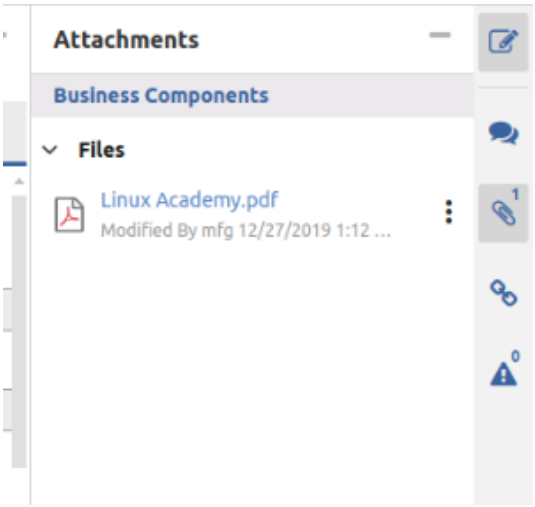
Show/hide the attachments panel

```
attachmentsPanel.IsVisible = true;
attachmentsPanel.IsVisible = false;
```

Get attachments data of the attachments panel

Get attachments data of the attachments panel

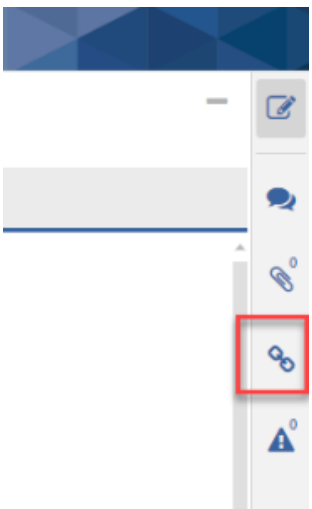
```
attachmentsPanel.getAttachments().then(attachments => console.log(attachments));
// Or the modern way
const attachments = await attachmentsPanel.getAttachments();
console.log(attachments)
```



Getting access to the drill-down panel

Getting access to the drill-down panel

```
const drillDownPanel = this.ViewController.SidePanel.DrillDownPanel;
```



Show/Hide the drill-down panel icon

Show/hide the drill-down panel

```
drillDownPanel.IsVisible = true;
drillDownPanel.IsVisible = false;
```

Show/Hide the drill-down link

Show/hide the drill-down link

```
drillDownPanel.setDrillLinkVisible('drillId', true);
drillDownPanel.setDrillLinkVisible('drillId', false);
```

Enable/Disable the drill-down link

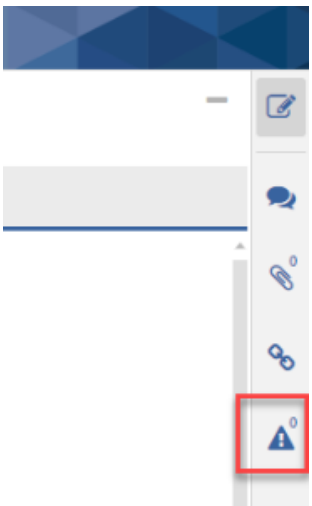
Enable/disable the drill-down link

```
drillDownPanel.setDrillLinkDisabled('drillId', false);
drillDownPanel.setDrillLinkDisabled('drillId', true);
```

Getting access to the message log panel

Getting access to the message log panel

```
const messageLogPanel = this.ViewController.SidePanel.MessageLogPanel;
```



Show/Hide the message log panel icon

Show/hide the log panel

```
messageLogPanel.IsVisible = true;
messageLogPanel.IsVisible = false;
```

Getting access to the message log object

Getting access to the message log object

```
const messageLog = this.ViewController.SidePanel.MessageLogPanel.MessageLog;
```

For more information about working with the message log, see Display message manager (DisplayMessageManager) #Manipulatethemessagelog

UI event handler development tips and tricks

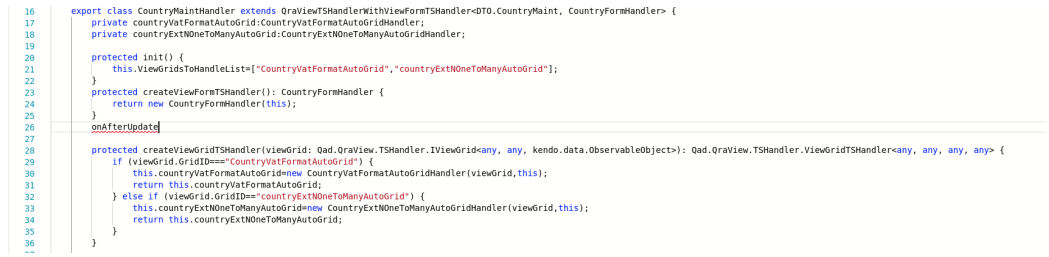
Introduction

This page describes some tips and tricks you can use when developing UI event handlers.

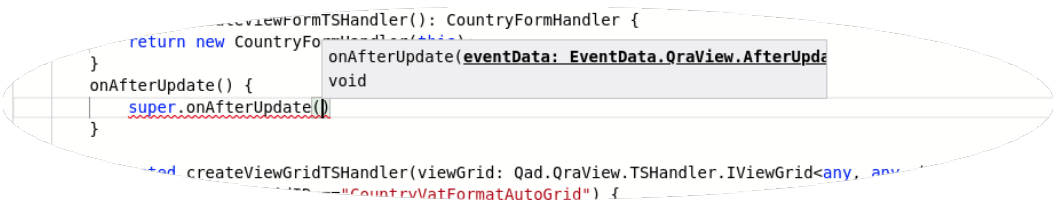
Overriding an event handler base class method

Knowing which methods to override can be easily done by typing "on" inside a class implementation, but outside a function, like this:

If you then select the function, it will add the function, but not its parameters:



The easiest way to get to the parameters is by completing the method without parameters, and then in the method body type super.OnAfterUpdate (the method you are overriding), and then the open bracket. By doing that, a popup will be shown with the parameters, and from that popup, you can select and copy the parameters:



These parameters can then be pasted to complete the method. This video shows all the steps:

Your browser does not support the HTML5 video element

Finding the ID of UI elements

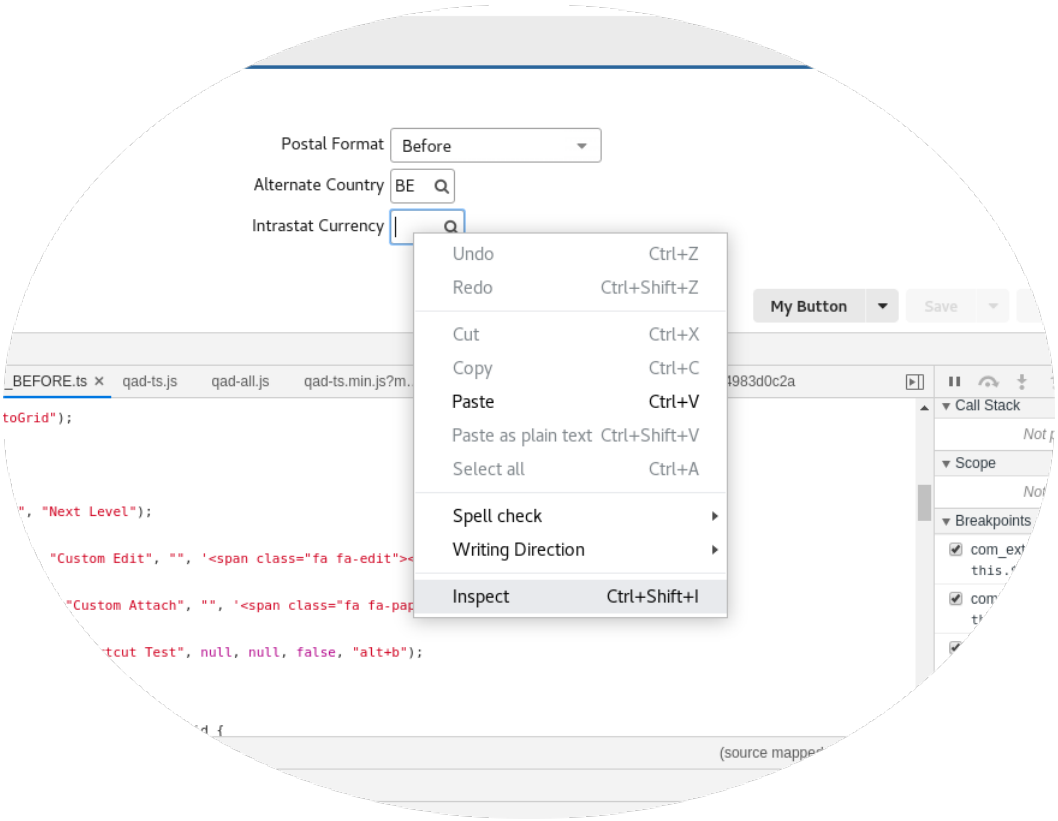
Finding the ID of a specific UI element, such as a form field or a grid can be done in a few ways. The first way is to use the constants you have available in your editor:

```
let vatGrid=this.ViewController.getViewGrid(Constants.DataGridNames.CountryVatFormatAutoGrid);
```

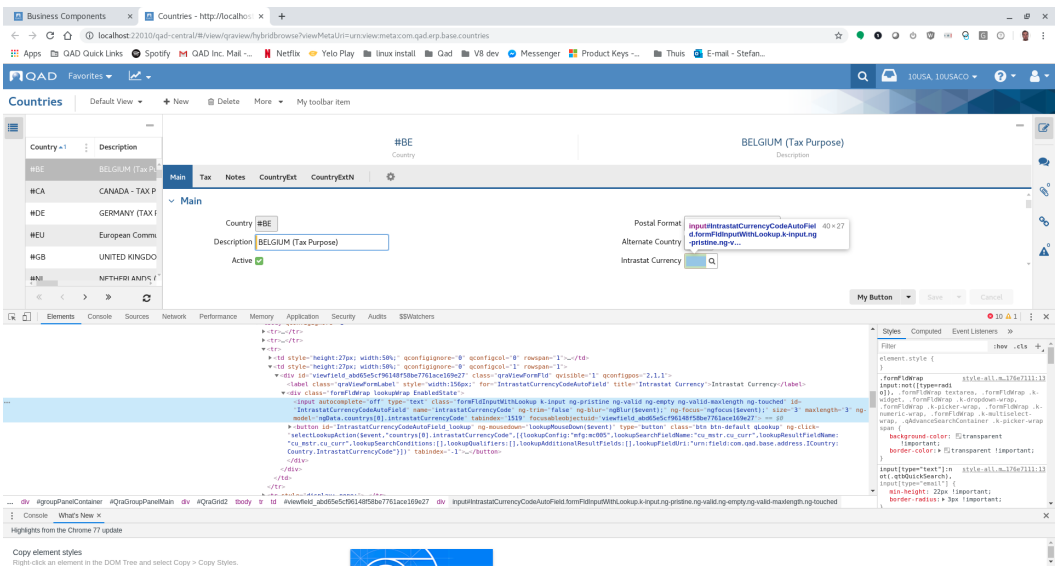
However, the Constants are only available for elements that belong to the UI itself, and not the ones coming from an embedded Business Component. So, if you extend Country, for example, with CountryExtNOneToMany, you won't see the grid that is added by this embedded BC. Also if the relationship is One-to-one, you won't see the fields of that embedded BC.

In order to find the ID of such an element, you can look into the HTML to find the id. This is a method that works for all elements. Do the following steps:

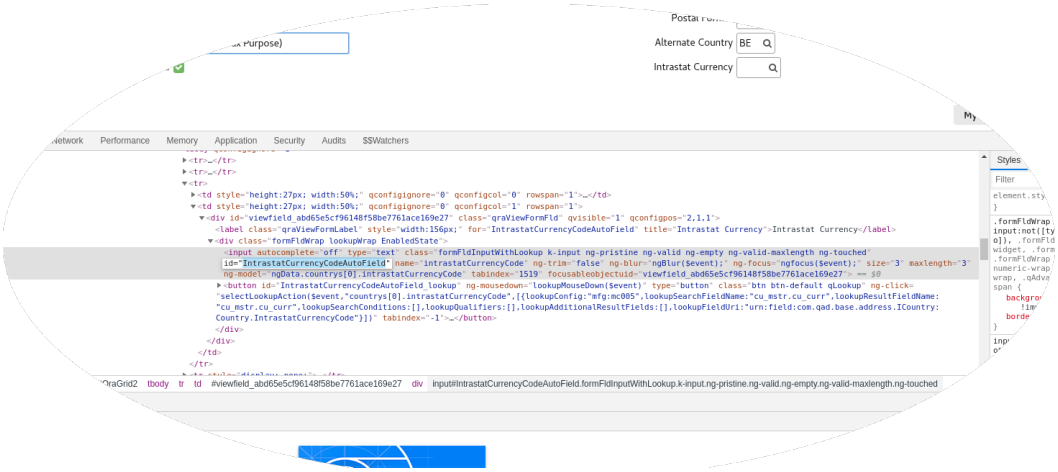
1) Open the UI in question and point to the element you want to see the ID for. Then, right-click:



2) Select Inspect, this will bring you to the HTML of that element:



If you look to that html, you will see the ID property:



3) Copy the ID and use it in your code.

This video demonstrates all the steps to get the ID of a field:

Your browser does not support the HTML5 video element

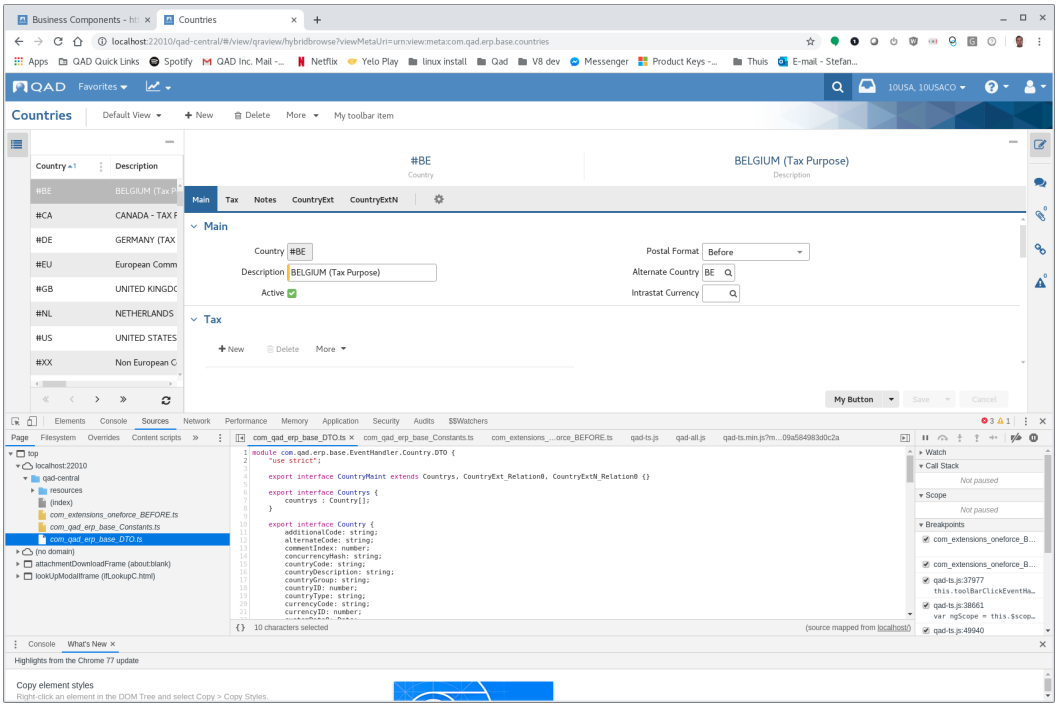
And this video shows how to do that for a grid. Note that for a grid, you need to remove the kGrid_ prefix from the ID:

Your browser does not support the HTML5 video element

Finding the correct grid DTO type

When creating a grid event handler, it is not always clear which DTO types to use in the class declaration. The different DTO types can be seen in the editor if you type "DTO." :

You can recognize the different grids in the data set and table names, but it might be easier to see the full declaration of these types. They can be seen if you go to the Countries screen, and then open the developer tools Sources tab and select the DTO file at the left:



You will see the different data sets and tables:

```
module com.qad.erp.base.EventHandler.Country.DTO {
    "use strict";

    export interface CountryMaint extends Countrys, CountryExt_Relation0, CountryExtN_Relation0 {}

    export interface Countrys {
        countrys : Country[];
    }

    export interface Country {
        additionalCode: string;
        alternateCode: string;
        commentIndex: number;
        concurrencyHash: string;
        countryCode: string;
        countryDescription: string;
        countryGroup: string;
        countryID: number;
        countryType: string;
        currencyCode: string;
        currencyID: number;
        customDate0: Date;
        customDate1: Date;
        customDate2: Date;
        customDate3: Date;
        customDate4: Date;
        customDecimal0: number;
        customDecimal1: number;
        customDecimal2: number;
        customDecimal3: number;
        customDecimal4: number;
        customInteger0: number;
        customInteger1: number;
        customInteger2: number;
        customInteger3: number;
        customInteger4: number;
        customLong0: string;
        customLong1: string;
        customNote: string;
        customShort0: string;
        customShort1: string;
        customShort10: string;
        customShort11: string;
        customShort12: string;
        customShort13: string;
```

```
        customShort14: string;
        customShort15: string;
        customShort16: string;
        customShort17: string;
        customShort18: string;
        customShort19: string;
        customShort2: string;
        customShort3: string;
        customShort4: string;
        customShort5: string;
        customShort6: string;
        customShort7: string;
        customShort8: string;
        customShort9: string;
        dataOperation: string;
        fiscalCountryCode: string;
        intrastatCurrencyCode: string;
        isActive: boolean;
        isDEACountry: boolean;
        isEUCountry: boolean;
        isGATTCountry: boolean;
        isInList: boolean;
        isNAFTACountry: boolean;
        lastModifiedDate: Date;
        lastModifiedUser: string;
        postalFormat: string;
        countryVatFormats: CountryVatFormat[];
    }

    export interface CountryVatFormat {
        concurrencyHash: string;
        countryCode: string;
        countryID: number;
        countryVatFormatID: number;
        dataOperation: string;
        lastModifiedDate: Date;
        lastModifiedTime: number;
        lastModifiedUser: string;
        taxFormat: string;
    }

    export interface CountryExt_Relation0 {
        CountryExt_Relation0: CountryExt[];
    }

    export interface CountryExt {
        countryCode: string;
        extraInfo: string;
    }
    export interface CountryExtN_Relation0 {
        CountryExtN_Relation0: CountryExtN[];
    }

    export interface CountryExtN {
        countryCode: string;
        extraInfoN: string;
        lineNr: number;
    }
}
```

For grids, you always need the main data set type that combines all the others (in this case, CountryMaint), and then the correct grid table record (this is always an interface with field variables only, e.g. CountryExtN).

And this is how you come to, for example, this class declaration:

export class CountryExtNOneToManyAutoGridHandler extends Qad.QraView.TSHandler.ViewGridTSHandlerV2<DTO.CountryMaint,DTO.CountryExtN[],DTO.CountryExtN> {

The second generics parameter is always an array of this record type, third parameter is the record type.

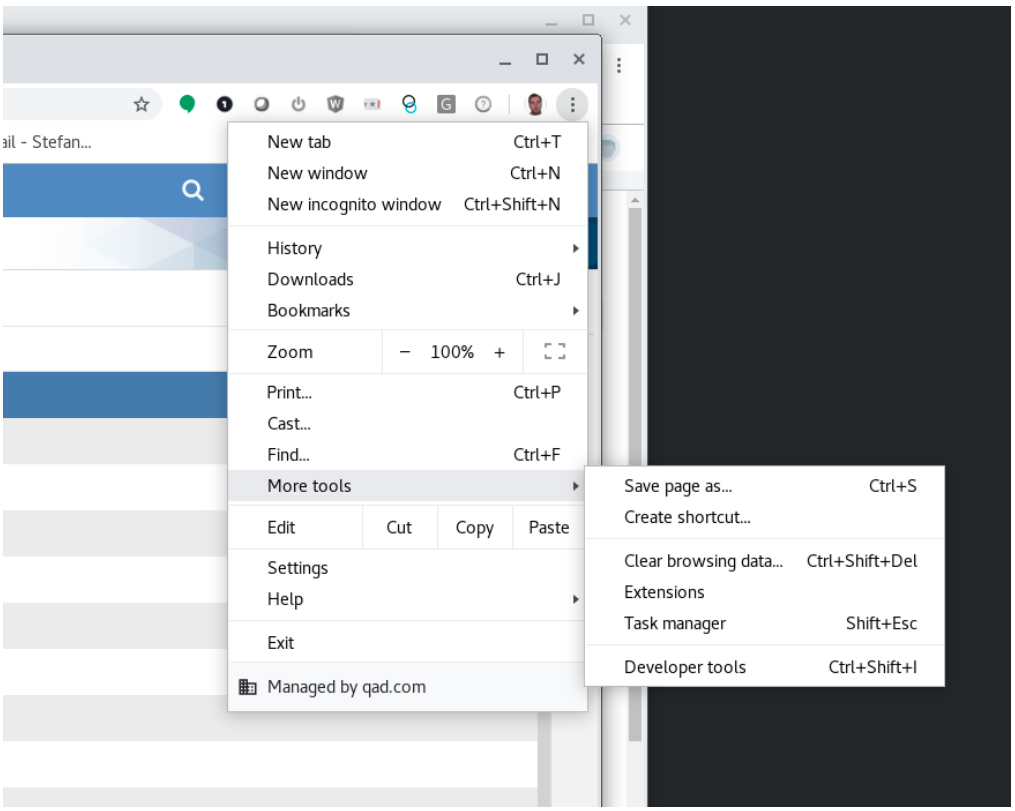
Debugging UI event handlers

Introduction

This page describes how to debug event handlers using the Chrome developer tools.

Chrome developer tools

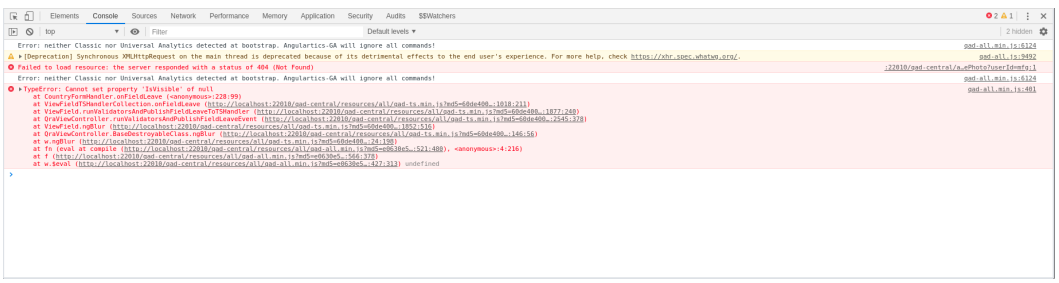
In order to debug a UI event handler, the first thing you need is the Chrome developer tools. You can open the developer tools here:



Or with the following key combination: Ctrl+Shift+I.

The developer tools are explained in depth here: https://developers.google.com/web/tools/chrome-devtools/.

The most important tabs for us are the Console:



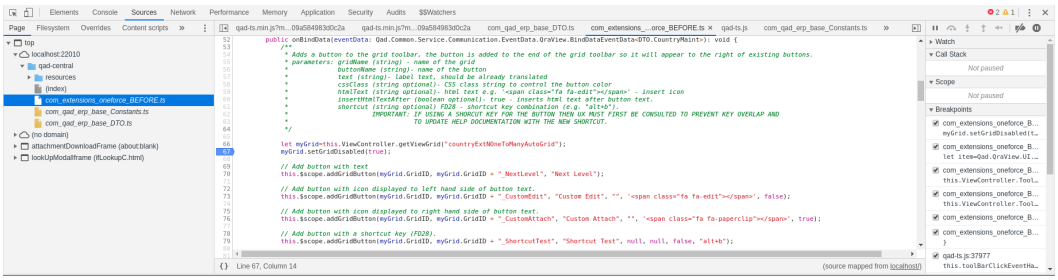
And the Sources:

As you can see, on the Console, you can see errors; click the stack trace lines to go to source code.

Opening a UI event handler from the Sources tab

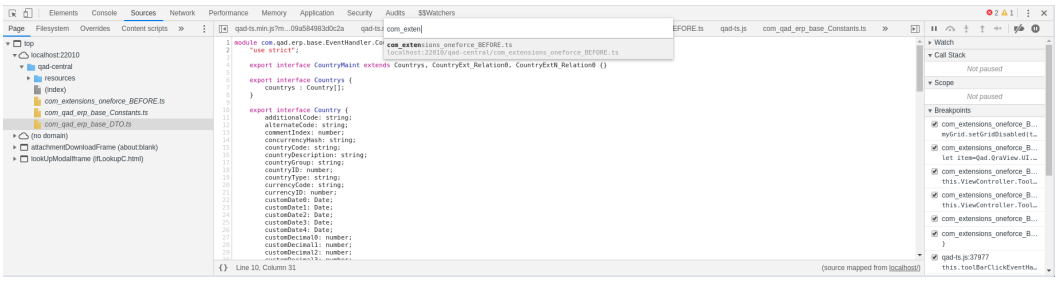
You can open your event handler in two ways:

1) Navigate to it:



As you can see at the left, you can browse through the page script sources. Your event handler will start with your app uri, but ':' replaced with . So, in this example, the app uri is urn:app:com.extensions.oneforce, the name of the UI event handler sources is com_extensions_oneforce_TIMING.ts. Timing can be BEFORE, AFTER, or PRIMARY.

2) Open it directly with ctrl-o, and then type part of the name of your UI event handler:



After typing part of the name, you can select your handler from the drop-down list.

As you can see, at the left, you can also open the DTO sources, that can be handy to check what's in there.

Adding breakpoints to your handler

Here, we again have two options:

You can click the line number:

video:

Your browser does not support the HTML5 video element

Or add a debugger statement to your code:

```
let myGrid=this.ViewController.getViewGrid("countryExtNOneToManyAutoGrid");
debugger;
myGrid.setGridDisabled(true);
```

The second is easier if you don't want to open the source file manually, because of the debugger line, the source file will automatically be opened when the line is hit.

Logging to the console

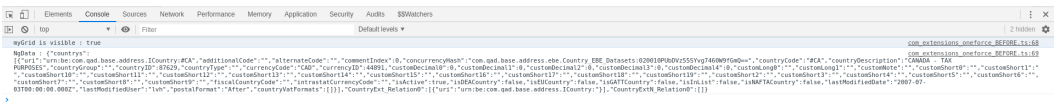
Another way of debugging is logging messages to the console with console.log:

```
let myGrid=this.ViewController.getViewGrid("countryExtNOneToManyAutoGrid");
myGrid.setGridDisabled(true);
console.log("myGrid is visible : " + myGrid.IsVisible);
```

If you want to log DTO's, it's best to convert them to a string first as follows:

```
console.log("NgData : " + JSON.stringify(this.NgData));
```

Example output:



Server scripting using TypeScript

A new feature is developed in QAD Adaptive ERP that allows a user to extend the business logic by applying server-side scripting in TypeScript. There are hooks made available in the Progress Code that can run TypeScript extensions made by the user.
This section describes how to setup, develop and deploy this server-side scripting

Development can be done on a "development machine" which can be windows or UNIX. To develop you need to setup a development environment so the TS code you write can be compiled and deployed

Deployment is done to the "QAD Adaptive ERP database" (called as server) to the table "ServerScript" where it is picked up and injected in the Progress Layer.

In this space, you will find 2 sections: one that describes enabling a development matchine for TS development (which has to be done once), and one that has step-by-step instructions to develop specific code and get this deployed to the server.

Setting up a server scripting development environment

Introduction

These pages describe in general how to setup a development environment for server scripting. What needs to be done is as follows:

1. Install the TypeScript compiler and Visual Studio Code.
2. Install command line tools: these tools are needed to perform the other steps described here.
3. Install the TypeScript api on the server: this step downloads the latest version of the TypeScript API from subversion and installs it on the server.
4. Generate, compile, and upload proxies for Business components you want to use in server scripts: in order to extend BCs with scripts, or to call other BCs in your scripts, TypeScript proxy files need to be generated for these BCs.
5. Create a TypeScript server script development project structure: create the necessary file structure to be able to start developing scripts with Visual Studio Code.
6. Compile your project and upload it to the server: the last step necessary to install and run your server scripts.

Detailed explanation:

Setup-1: Configuring the TypeScript compiler and Visual Studio Code On a PC4 Windows VM (Version 4.3.1)

Table of Contents:

Introduction

This page describes how to configure a Windows VM for TypeScript development with Visual Studio Code.
Note that the Windows VM is a Windows server 2016. That's why some steps are different from doing this on a Windows 10 environment.
This page assumes that you are working on a Windows VM, and that you are logged in with your QAD user.

Steps

Step 1: Configure the TypeScript compiler (tsc)

The TypeScript compiler is a JavaScript script that runs from **NodeJS**. NodeJS uses a package manager called **npm** that needs to be used to install TypeScript. NodeJS and npm are already installed on the Windows VM. The TypeScript compiler is not.

1- Check nodeJS on command prompt:

```
node --version
```

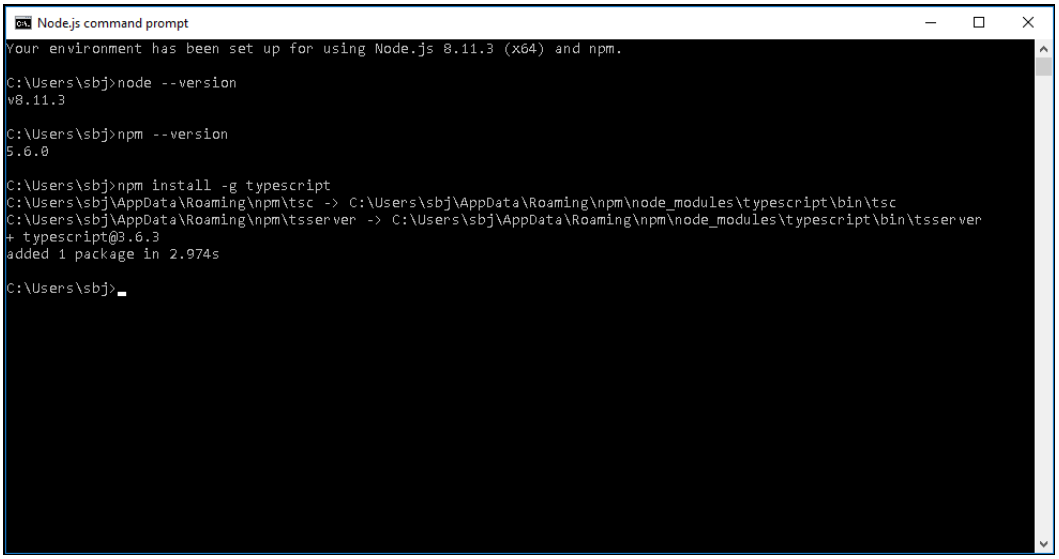
2- Check npm on command prompt:

```
npm --version
```

3- install the TypeScript compiler on command prompt::

```
npm install -g typescript
```

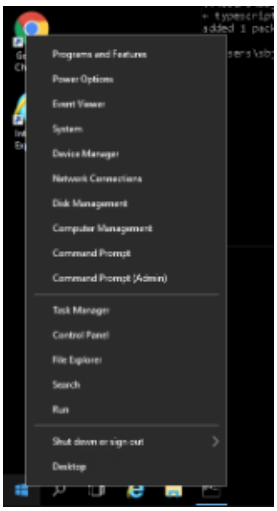
You should see the following output for these 3 commands:



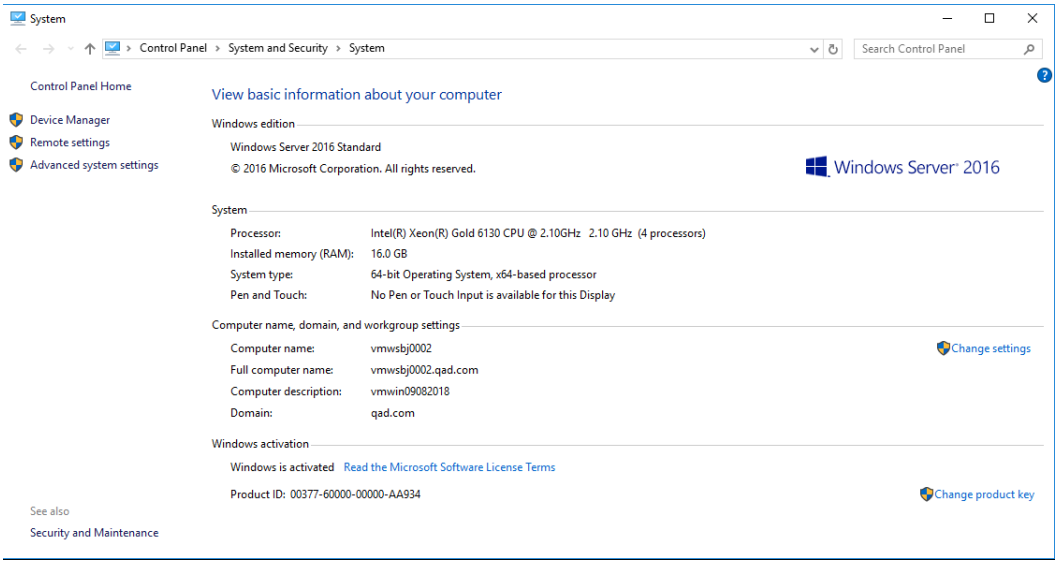
Step 2: Configure the PATH environment variable

On a Windows Server, the PATH environment variable is not automatically adjusted with the path to tsc. So you need to do the following steps to adjust it:

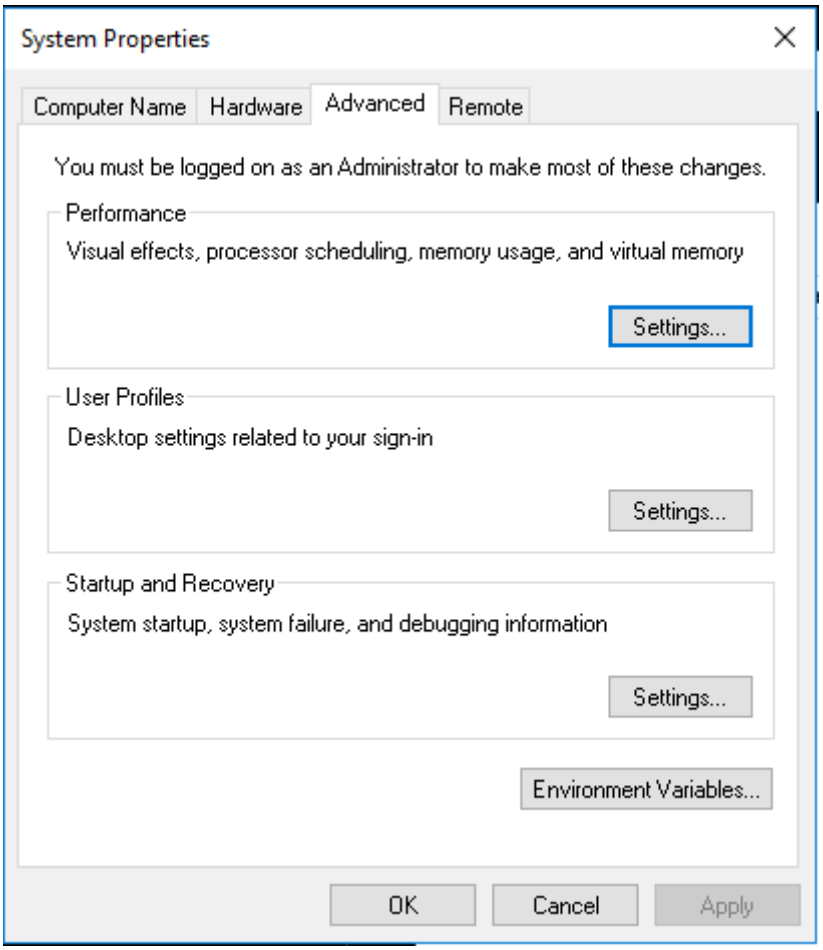
1- To open the System setting, right-click the **Start** button and select **System**:



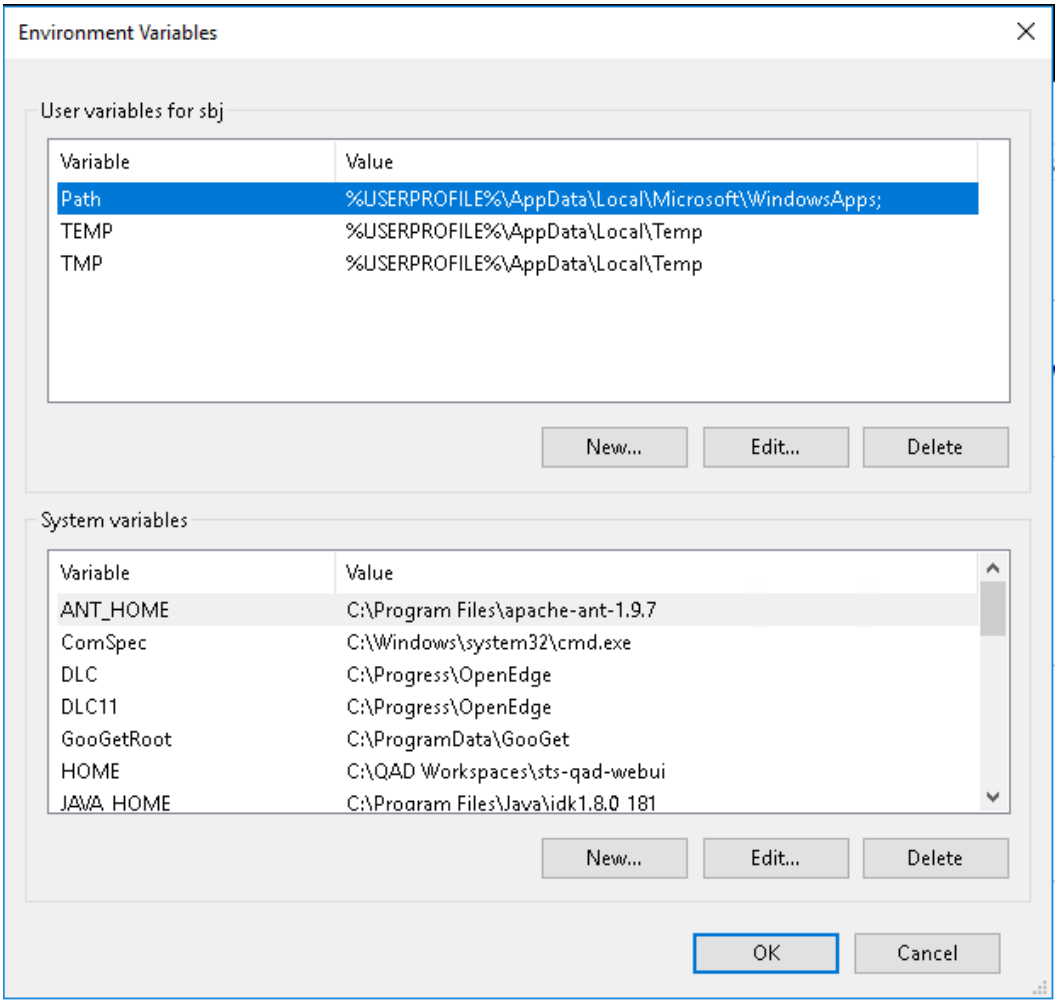
2- It will open the following window:



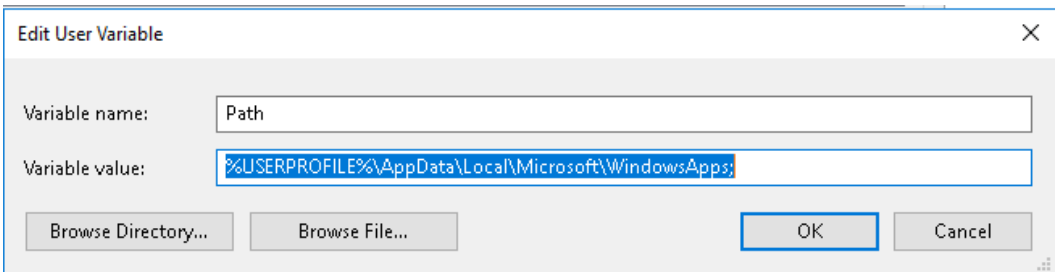
3- Then, click the **Advanced** system settings. It will open the following dialog box:



4- Next, click **Environment Variables**.



5- Make sure the Path is selected in the User variables, and then click **Edit** (the one for user variables).



6- This is the place where you need to add the path to tsc. In this case, this is C:\Users\sbj\AppData\Roaming\npm. However, it's better to specify the path in a more generic way like this: **%USERPROFILE%\AppData\Roaming\npm** Add this path to the variable value, after the ;

7- Click OK on all the windows until everything is closed and exit the dos command window.

Step 3: Test tsc

1- Open dos command again and run 'tsc --version' to validate if it works.

```
tsc --version
```

Video

The following video shows all the steps:

Your browser does not support the HTML5 video element

After this, your Windows VM is ready for TypeScript development.

Setup-2: Installing the command line tools

Table of Contents:

Introduction

A server script always extends an existing Business Component. A possible extension could be to call other Business Components. All these BCs all have a Progress implementation. Therefore we need to make it possible to call TS from a Progress BC on the one hand, and to call a Progress BC from TS on the other hand. That is done through what we call **TypeScript Proxies**. These proxies are **Type Definition** files that represent JavaScript files that can communicate with the progress implementation.

These proxies are currently not built automatically by the build script of the different apps. That means that we need to do that manually in our development environment, and after building them, we need to upload them to the server, and copy the type definition files to our development project. Also, the server scripts that we develop need to be compiled and uploaded to the server in order to run them.

In order to do all this, we need to set up our development environment with tools and scripts to do this from the command line. This page describes how to do that.

Characteristics of Development Environment

A development environment for server scripting needs a running QAD adaptive erp installation. That is what we call the server in this documentation.

Currently, a development environment for server scripting has three important parts:

1. **The server TypeScript api**
   These are the JavaScript and type definition files that are the API for developing server scripts. It contains the base classes for the script and proxies as well as all the code that communicates with the Progress code.
2. **The App TypeScript proxies**
   These are also JavaScript and type definition files that represent the Business Components and data sets in an App. They exist for both standard QAD apps as for Platform apps (also the ones you developed yourselves).
3. **The developed server scripts**
   These are the scripts you will develop for your project. These scripts depend on #1 and #2 in this list.

For these three parts, we have to do the following steps to make our development environment ready.

Steps

Step-1: Installing and adapting the command line tools

The first thing we need to do is download and unpack the zip file that contains the TypeScript API, and install that API on the development matchine. (Note there is a zip available for Windows and UNIX. As in this exercise the development machine is Windows we download the windows zip).

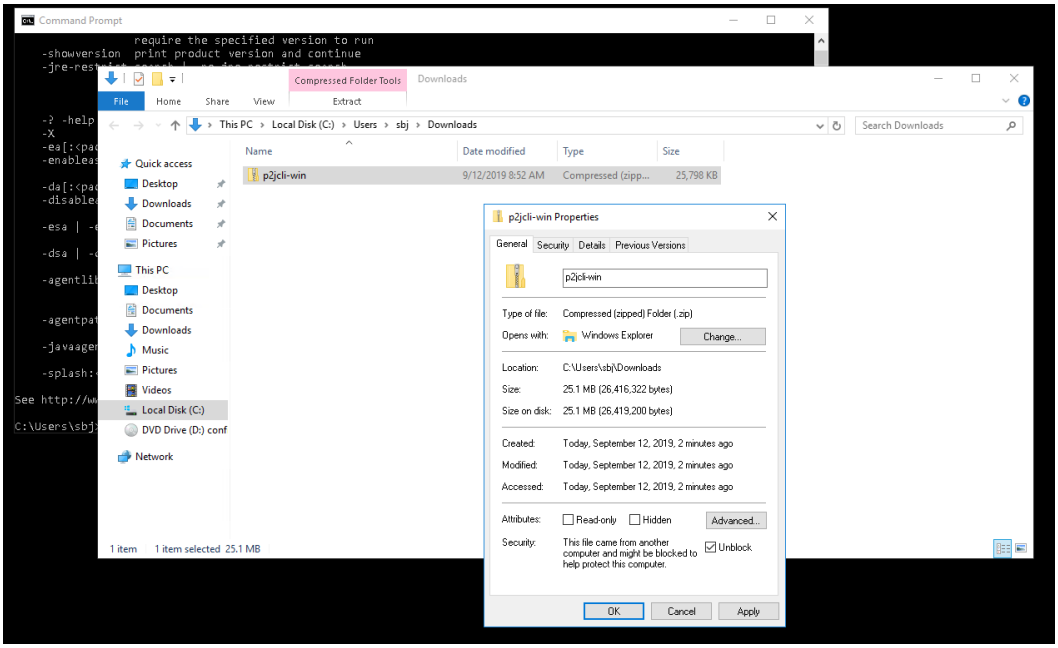
Project Root Folder

Your "project root folder" is a folder from your choice. In this exercise, we use C:\Users\sbj\develop\sss (or %HOMEPATH%\develop\sss)

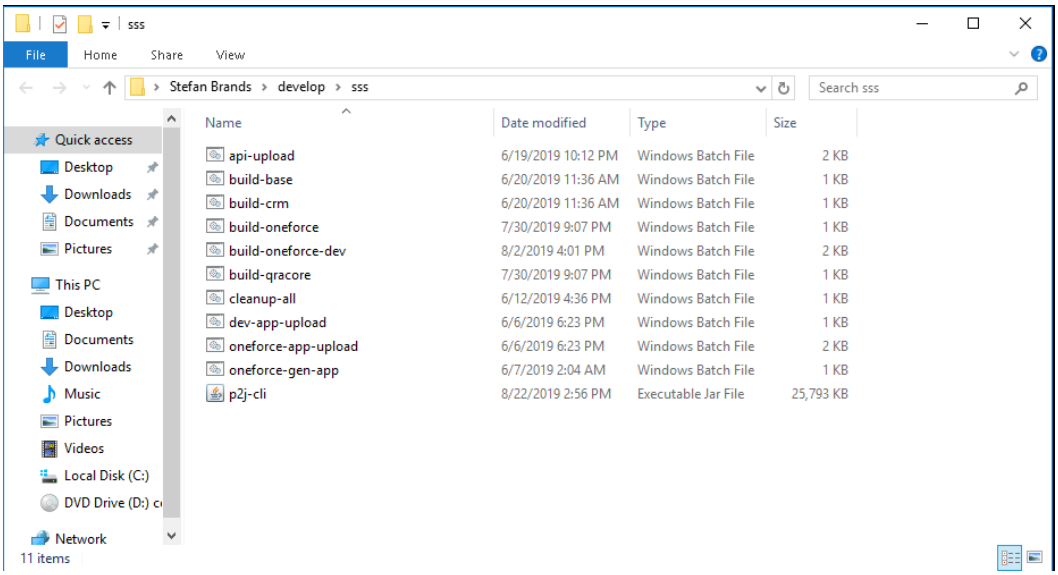
1- Download and unzip the zip file in the root of your development folder. They can be downloaded from here:

- Windows: http://subversion.qad.com:18080/svn/qfs_repository/modules/qracore/sss/trunk/p2jcli/artifact/p2jcli-win.zip
- Linux: http://subversion.qad.com:18080/svn/qfs_repository/modules/qracore/sss/trunk/p2jcli/artifact/p2jcli-unix.zip

2- When downloading on Windows, make sure you remove the "This file came from another computer ..." block by right-clicking the zip file and selecting the **Unlock** checkbox, and then click **OK**.



After unzipping, you'll see the following files.



3- Adapt all the bat files to connect to the correct server. We take the example of the api-upload.bat source code. You can see the line at the top that sets the **backendURL** needs to be changed to your server.

```
@echo off
rem this script is used for fetching the latest version of api files from svn

set backendURL="https://vmlsbj0002.qad.com:22011/qad-central/"

set DESTINATION=tmp
set CURRENT_DIR=%CD%

set REPO_URL=http://subversion.qad.com:18080/svn/qfs_repository/modules/qracore/sss/trunk
set API_PATH=typescript/api/dist/api.d.ts
set API2_PATH=typescript/api/dist/api.js
set API3_PATH=typescript/api/dist/api.js.map
set P2JS_PATH=typescript/p2js/dist/p2js.d.ts
set P2JS2_PATH=typescript/p2js/dist/p2js.js
set P2JS3_PATH=typescript/p2js/dist/p2js.js.map
```

```
if not exist %DESTINATION% mkdir %DESTINATION%

cd %DESTINATION%

echo "Connecting to svn..."
svn export --force %REPO_URL%/%API_PATH%
svn export --force %REPO_URL%/%P2JS_PATH%
svn export --force %REPO_URL%/%API2_PATH%
svn export --force %REPO_URL%/%P2JS2_PATH%
svn export --force %REPO_URL%/%API3_PATH%
svn export --force %REPO_URL%/%P2JS3_PATH%

echo "Starting upload..."

cd %CURRENT_DIR%

echo "Uploading..."
java -jar p2j-cli.jar --backend.url=%backendURL% "upload -a urn:app:com.qad.qracore -aseq 0 -fseq 0 -fn %
DESTINATION%/p2js"
java -jar p2j-cli.jar --backend.url=%backendURL% "upload -a urn:app:com.qad.qracore -aseq 0 -fseq 1 -fn %
DESTINATION%/api"

cd %DESTINATION%
del api.d.ts
del p2js.d.ts
del api.js
del p2js.js
del api.js.map
del p2js.js.map

cd %CURRENT_DIR%
```

This needs to be done for all the bat files or shell scripts that you will be using for your project.

The different files explained

The heart of the tools is the **p2j-cli.jar** Java program that has all the logic to communicate with the QAD adaptive erp installation on the server.

By default, it uses mfg with a blank password to authenticate to the server. If you need to change that, you need to add -auth.user=<User> --auth.pass=<Password> to all the calls to plj-cli.jar like in this example:

```
java -jar p2j-cli.jar --auth.user=youruserid --auth.pass=yourpwd --backend.url=%backendURL%
"upload -a urn:app:com.qad.qracore -aseq 0 -fseq 0 -fn %DESTINATION%/p2js"
java -jar p2j-cli.jar --auth.user=youruserid --auth.pass=yourpwd --backend.url=%backendURL%
"upload -a urn:app:com.qad.qracore -aseq 0 -fseq 1 -fn %DESTINATION%/api"
```

The rest of the files are all scripts that make use of p2j-cli.jar. The function of the files is:

- api-upload: retrieve the server TS api from subversion and upload it to the server
- build-qracore: generate proxies for the qracore app, build them and upload them to the server
- build-base: generate proxies for the base app, build them and upload them to the server
- build-crm: generate proxies for the crm app, build them and upload them to the server
- build-qadextensions: generate proxies for the qadextensions app, build them and upload them to the server
- build-qadextensions-dev: build and upload the scripts developed for the qadextensions app
- build-oneforce: generate proxies for the oneforce app, build them and upload them to the server (note, this app is not installed by default)
- build-oneforce-dev: build and upload the scripts developed for the oneforce app (note, this app is not installed by default)
- cleanup-all: delete all scripts from the server (except the api)

Step-2: Modifying the scripts for your project

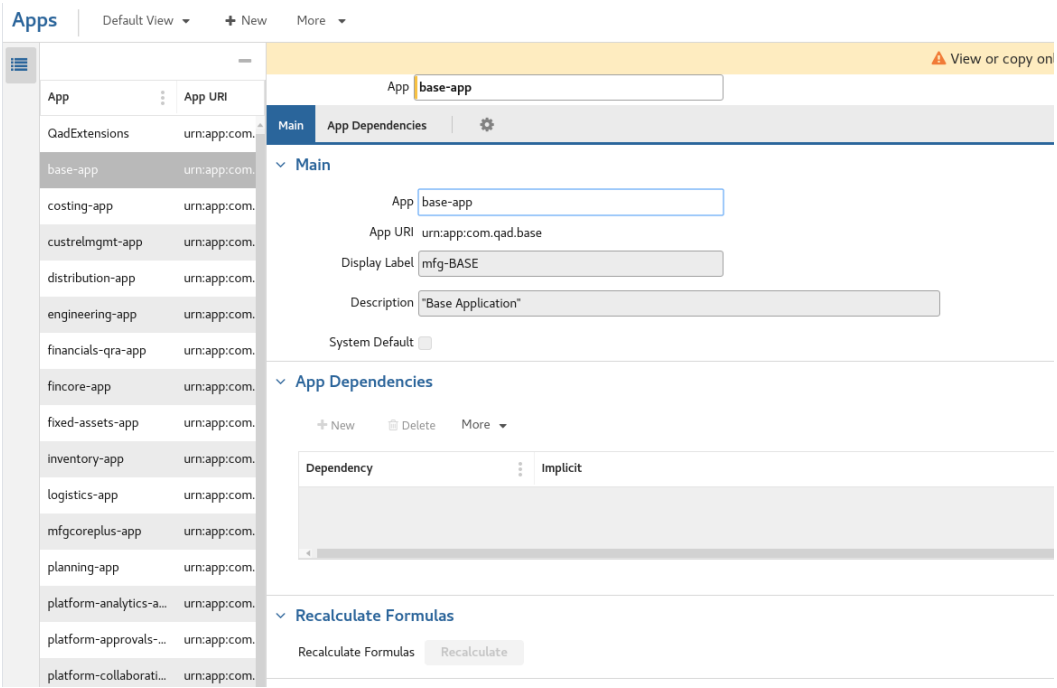
As you can see in the list of files above, there are scripts for generating proxies for different apps. These scripts are only necessary if you want to extend or call the business components of these apps. You also see that there are no scripts for all apps in the system. So, if you need other apps, you need to create scripts for them.

1- Creating a proxies build script for another app

The best way to do this is to copy the build-qadextensions script and modify the following section at the top:

```
set backendURL="https://vmlsbj0002.qad.com:22011/qad-central/"
set appURI=urn:app:com.extensions.qadextensions
set appFolder=QadExtensions
set appDistName=qadextensionsgen
```

Modify four parameters to your needs. Note that the AppFolder name always starts with a capital letter and dashes and the "App" suffix are removed. So, "base-app":



Is converted to "Base".

appDistName is always all in lower case letters.

2- Modifying your app's dev scripts

As you can see in the list of files above, there are two scripts for the app you want to develop scripts for. In this example, there are scripts for two apps, namely the OneForce app and the QadExtensions app. We'll look at the QadExtensions as an example:

- build-qadextensions: generate proxies for the qadextensions app, build them and upload them to the server
- build-qadextensions-dev: build and upload the scripts developed for the qadextensions app

The first script is used to generate proxies for the business components you created (and deployed) in that app. The second script is used to build the server scripts you will be writing for this app.

These two scripts also need to be adapted, you need to add references to the other apps you are using in your app. For build-qadextensions, you need to add references to the other apps you are extending. For example, if you create a BC in your app that extends a BC from the base app (e.g. Country), then you need to add the base app to the script:

```
@echo off

set backendURL="https://vmlsbj0002.qad.com:22011/qad-central/"
set appURI=urn:app:com.extensions.qadextensions
set appFolder=QadExtensions
set appDistName=qadextensionsgen

set destinationFolder=gen

if not exist %destinationFolder% mkdir %destinationFolder%
rmdir %destinationFolder%\%appFolder% /S /Q

echo "Generating proxies..."
java -jar p2j-cli.jar --backend.url=%backendURL% --output.folder=%destinationFolder% "gen -a %
appURI%"

echo "copy app dependencies"

rem uncomment the next line if you are extending the Base app
```

```
rem copy %destinationFolder%\Base\dist\basegen.d.ts %destinationFolder%\%appFolder%\lib /Y
rem uncomment the next line if you are extending the QraCore app
rem copy %destinationFolder%\Qracore\dist\qracoregen.d.ts %destinationFolder%\%appFolder%\lib /Y


echo "Compiling proxies..."
call tsc -p %destinationFolder%/%appFolder%/tsconfig.json



echo "App has been compiled successfully. Storing app to database..."

echo "Looking for js files in dist folder..."

java -jar p2j-cli.jar --backend.url=%backendURL% --output.folder=%destinationFolder% "upload -a %
appURI% -aseq 3 -fseq 0 -fn %destinationFolder%/%appFolder%/dist/%appDistName%"

echo "App has been stored successfully."
```

For your convenience, the base app is already in the script, but it is commented out:

```
rem uncomment the next line if you are extending the Base app
rem copy %destinationFolder%\Base\dist\basegen.d.ts %destinationFolder%\%appFolder%\lib /Y
```

Uncomment the copy line, and add the same (modified) line for other apps you extend in your app.

For build-qadextensions-dev:

```
echo off

set backendURL=https://vmlsss0001.qad.com:22011/qad-central
set appURI=urn:app:com.extensions.qadextensions
set appFolder=QadExtensions
set appgenDistName=qadextensionsgen
set appdevDistName=qadextensionsdev

set destinationFolder=dev
set genFolder=gen

if not exist %destinationFolder% mkdir %destinationFolder%


echo "copy app dependencies"

copy %genFolder%\%appFolder%\lib\p2js.d.ts %destinationFolder%\%appFolder%\lib /Y
copy %genFolder%\%appFolder%\lib\api.d.ts %destinationFolder%\%appFolder%\lib /Y
copy %genFolder%\%appFolder%\dist\appgenDistName.d.ts %destinationFolder%\%appFolder%\lib /Y

rem uncomment the next line if you are extending the Base app
rem copy %genFolder%\Base\dist\basegen.d.ts %destinationFolder%\%appFolder%\lib /Y
rem uncomment the next line if you are extending the QraCore app
rem copy %genFolder%\Qracore\dist\qracoregen.d.ts %destinationFolder%\%appFolder%\lib /Y

echo "Compiling ..."
call tsc -p %destinationFolder%/%appFolder%/tsconfig.json


echo "App has been compiled successfully. Storing app to database..."

echo "Looking for js files in dist folder..."

java -jar p2j-cli.jar --backend.url=${backendURL} --output.folder=%destinationFolder% "upload -a %
appURI% -aseq 3 -fseq 1 -fn %destinationFolder%/%appFolder%/dist/%appdevDistName%"

echo "App has been stored successfully."
```

You need to add references to the same apps as for build-qadextensions AND for other apps that you want to call in your scripts. If you, for example, want to call a business component from QraCore, then you'll need to add a reference to the QraCore proxies here.

Next

After this, you are ready for the next step: Setup-3: Deploying TypeScript API to server (script: api-upload).

Setup-3: Deploying TypeScript API to server (script: api-upload)

Introduction

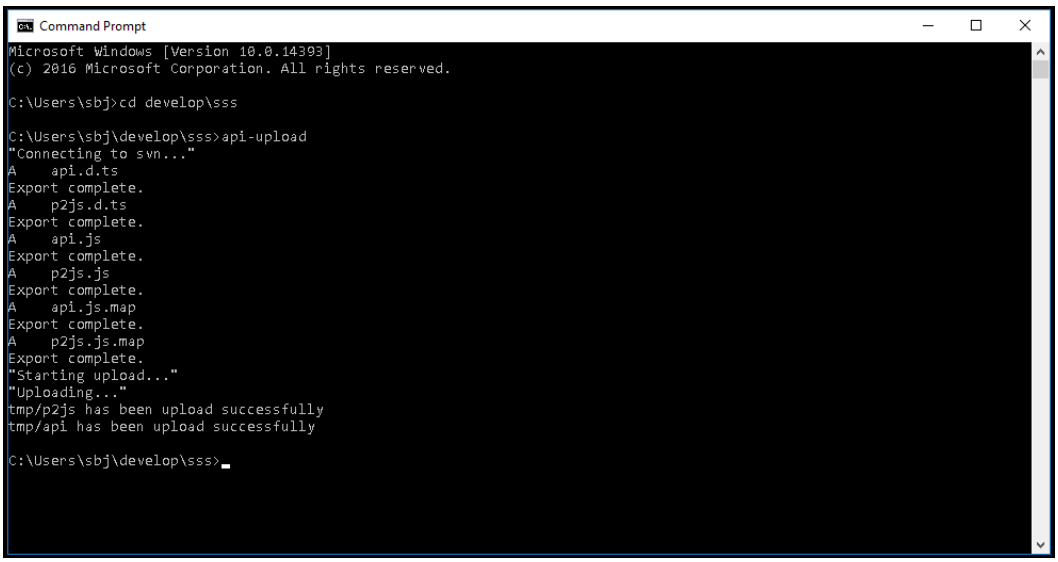
In the future, the server TypeScript API will automatically be installed with the QraCore package. However, at the moment it's a manual step that you can do by running the api-upload script. You can do that as follows:

Deploying the script

1. Open a dos prompt (if you are using a Windows VM, and your project folder is not under your home folder, then make sure that you open a dos prompt with administrator rights).
2. Then, cd to your project's root folder.
3. Execute the api-upload script.

Make sure you adapted api-upload script to your server as described here: Setup-2: Installing the command line tools

You should have the following output after doing the above steps.



After this, the server is ready for generating TS proxies. You can see how to do this on the next page: Generating, building, and deploying TypeScript proxies (script: build_<AppName>).

Setup-4: Generating, building, and deploying TypeScript proxies (script: build_<AppName>)

Introduction

Development of server scripts is done by extending business components with a script. That means that you need proxy classes for these business components. It also means calling business components of related apps. In order to be able to do that, you will also need the proxy files for the related apps these business components belong to. This page describes how to generate, build, and upload these proxies.

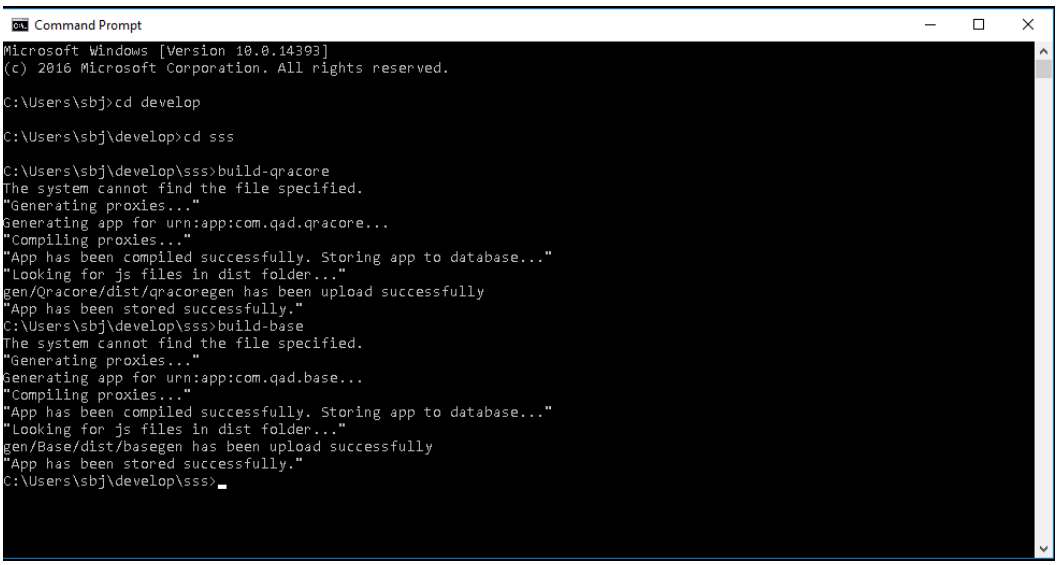
Generate related/dependent apps proxies

Because our development app depends on other apps, we first need to build the proxies for the dependent apps. This is currently a manual step, in the future, these proxies will be installed automatically by the app packages. Our scripts are already configured (see Setup-2: Installing the command line tools). So, all we need to do is to run the scripts that generate, build, and deploy these proxies. In this example, we generate proxies for Base and QraCore by running the following scripts:

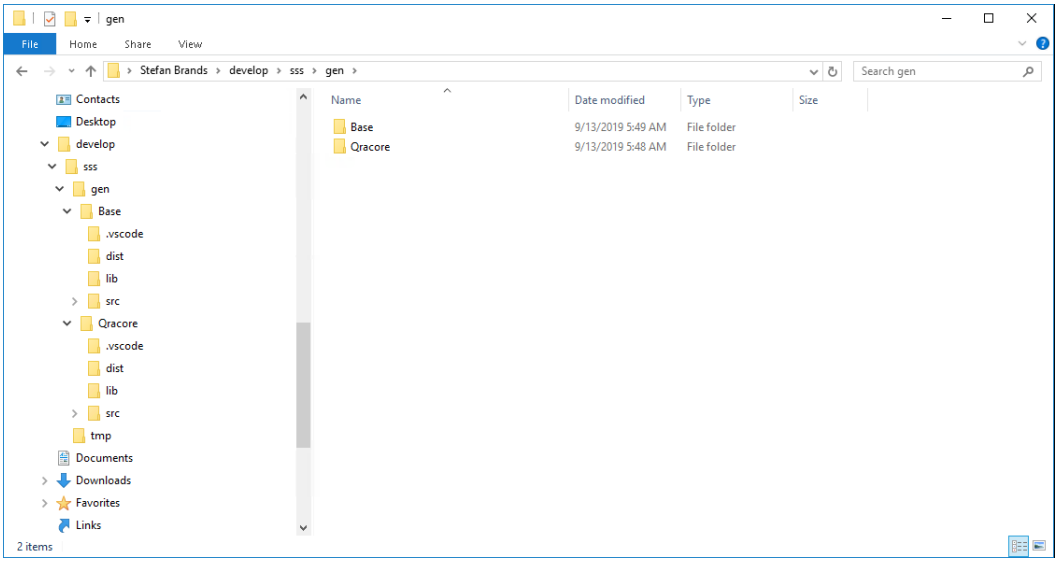
- build-qracore: generate proxies for the qracore app, build them and upload them to the server
- build-base: generate proxies for the base app, build them and upload them to the server

> Make sure you always run the scripts in the root folder of your project.

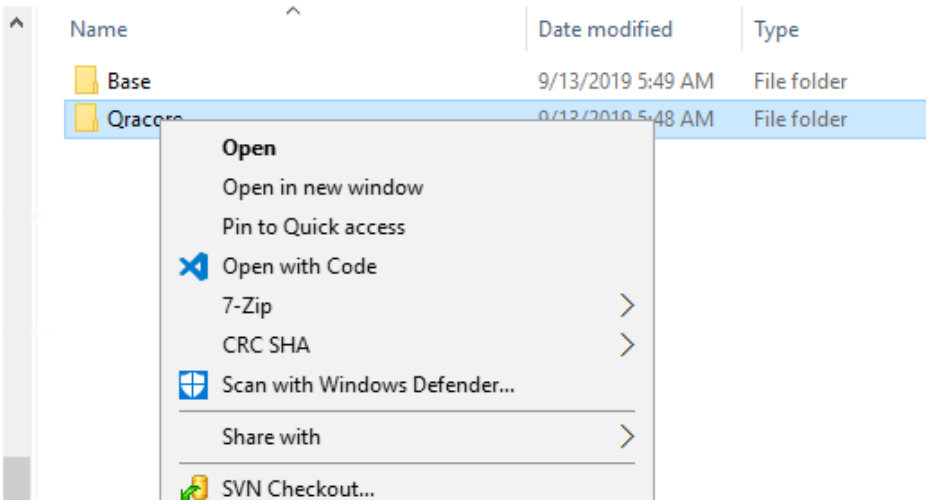
The output:



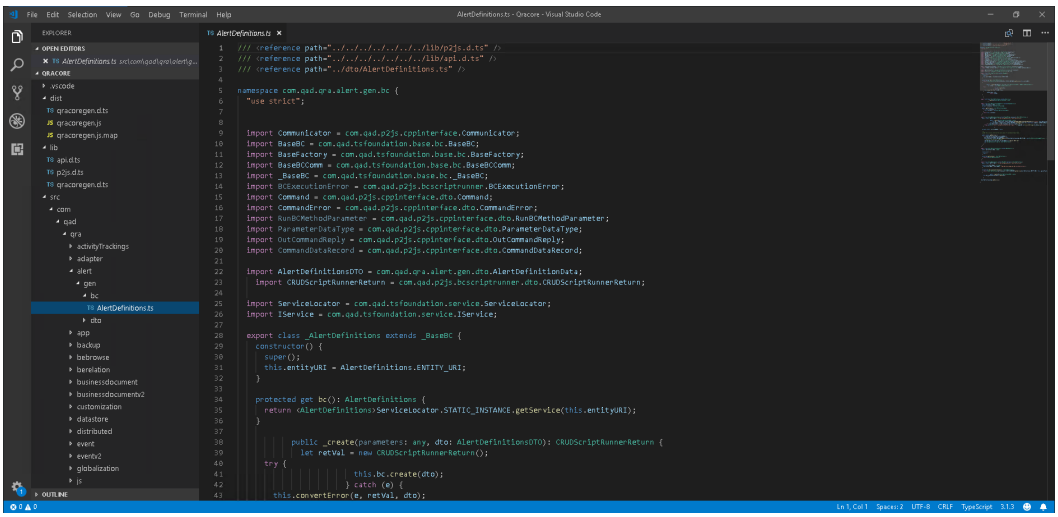
After executing the above steps, you'll see that there is a gen folder in your project folder, with two VS Code projects:



You can open these projects by right-clicking the QraCore, for example, and selecting **Open with Code**:

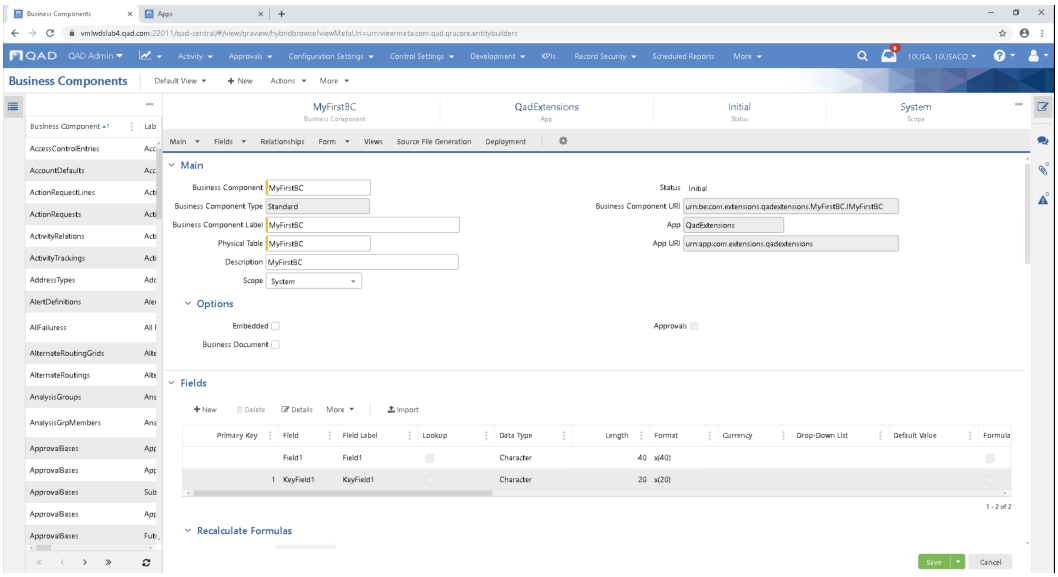


It will open the project for you in VS Code. You will find all the generated proxies in the src folder, and the build output in the dist folder. In the lib folder, you can see that the TS API is referenced:



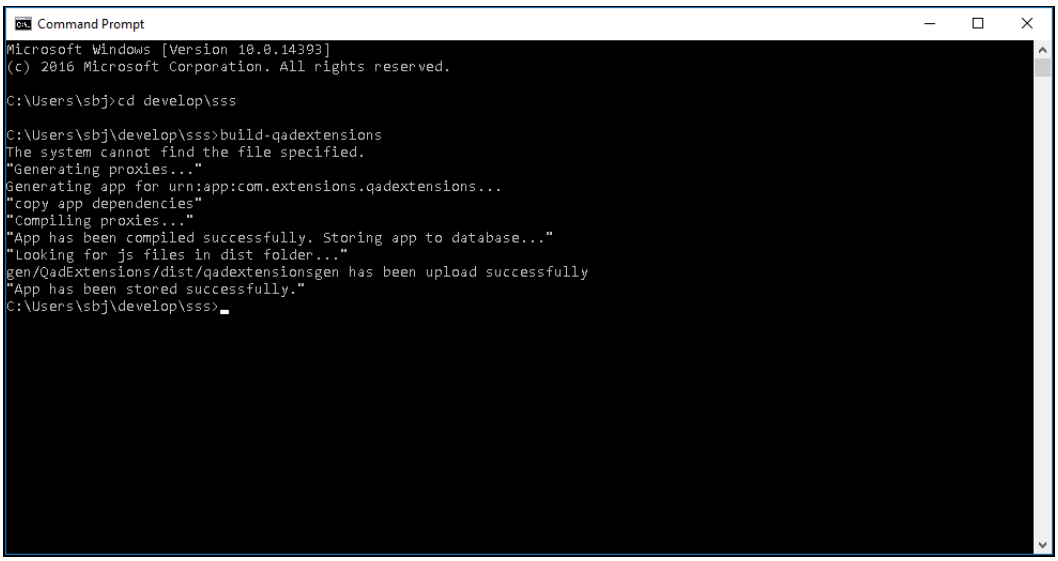
Generate your apps proxies

This section describes how to generate the proxies for your own app. Important here is that your app contains at least one Business Component. For the purpose of this example (also used on other pages in the area), it is best to create and deploy the following BC:



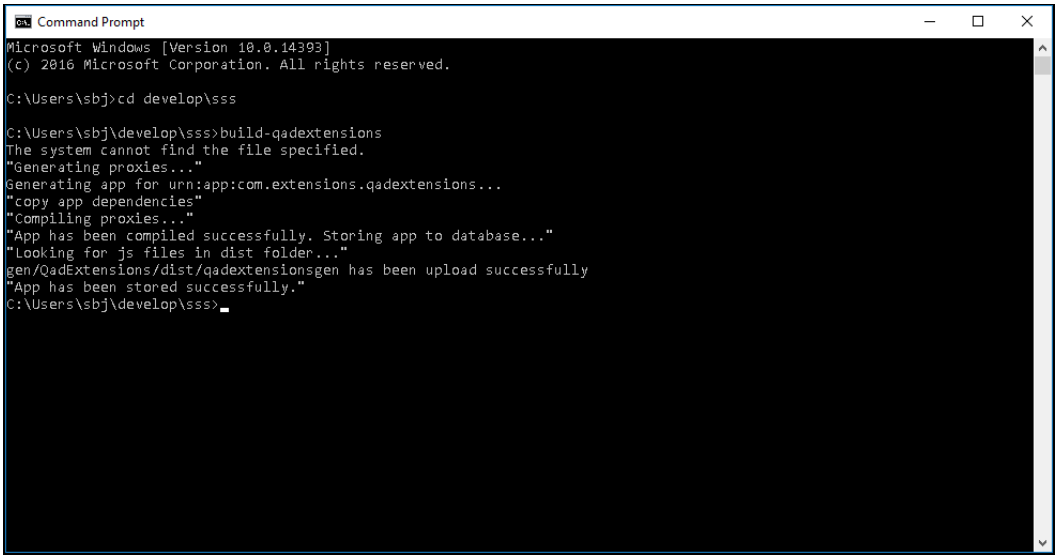
There must be at least one deployed Business Component in your app or this step will fail.

Now that you have a BC in your app, and proxies for the related apps, you can generate and build proxies for your own app. In this example, this is done for the QadExtensions app:

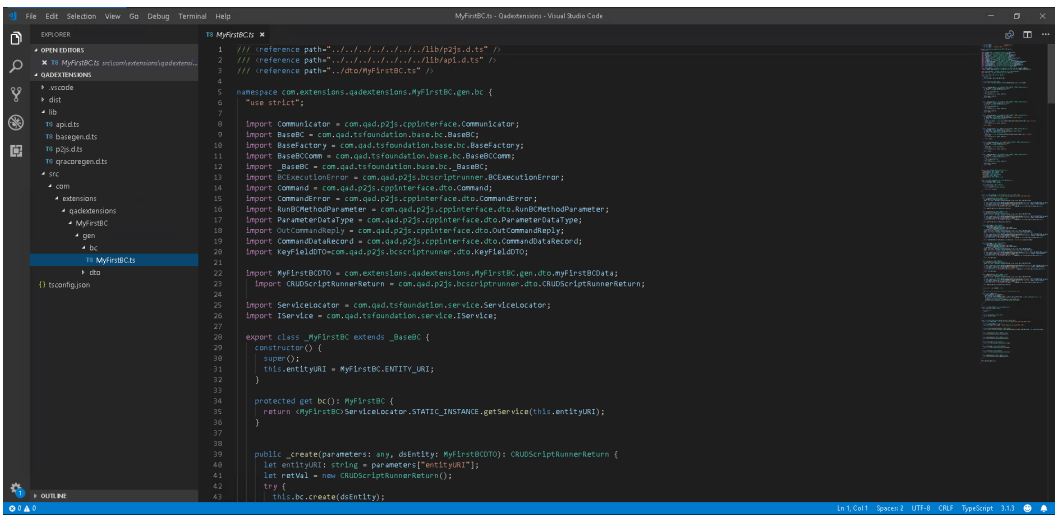


Make sure that the script for generating your proxies is adapted to include references to all apps you have extensions for as described on this page: Setup-2: Installing the command line tools

After doing this, you'll see an extra folder with the VS Code project for the proxies of your app:



If you open the VS Code project, you'll see proxies for the BCs you created in your app:



You can also see in the lib folder the related apps this proxies project includes, namely qracore and base.

After these steps, the proxies are ready to be used in a development VS Code project: Setup-5: Creating a Visual Studio Code project for developing scripts for your app.

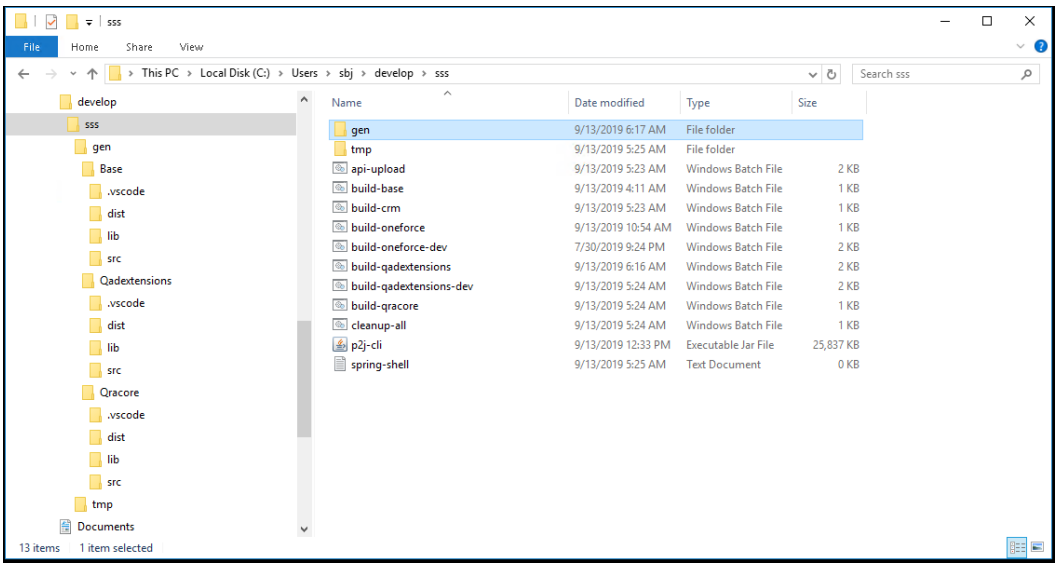
Setup-5: Creating a Visual Studio Code project for developing scripts for your app

Introduction

After generating all proxies, we can create our VS Code project to start developing our server scripts. This page describes how to create the project structure for this.

Step 1: reate the project directory structure and config file

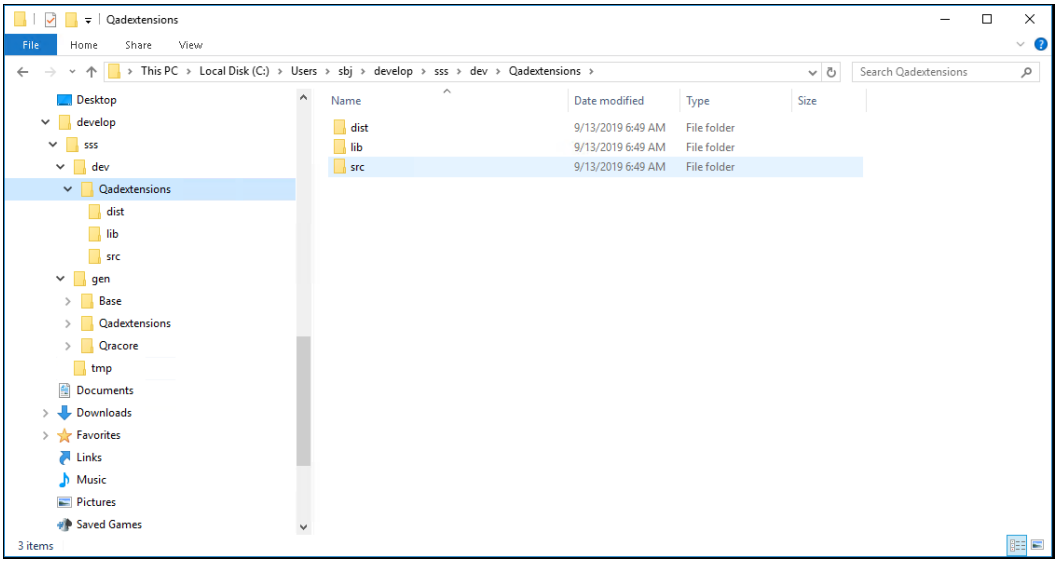
After following the previous steps, you should see a directory structure like this:



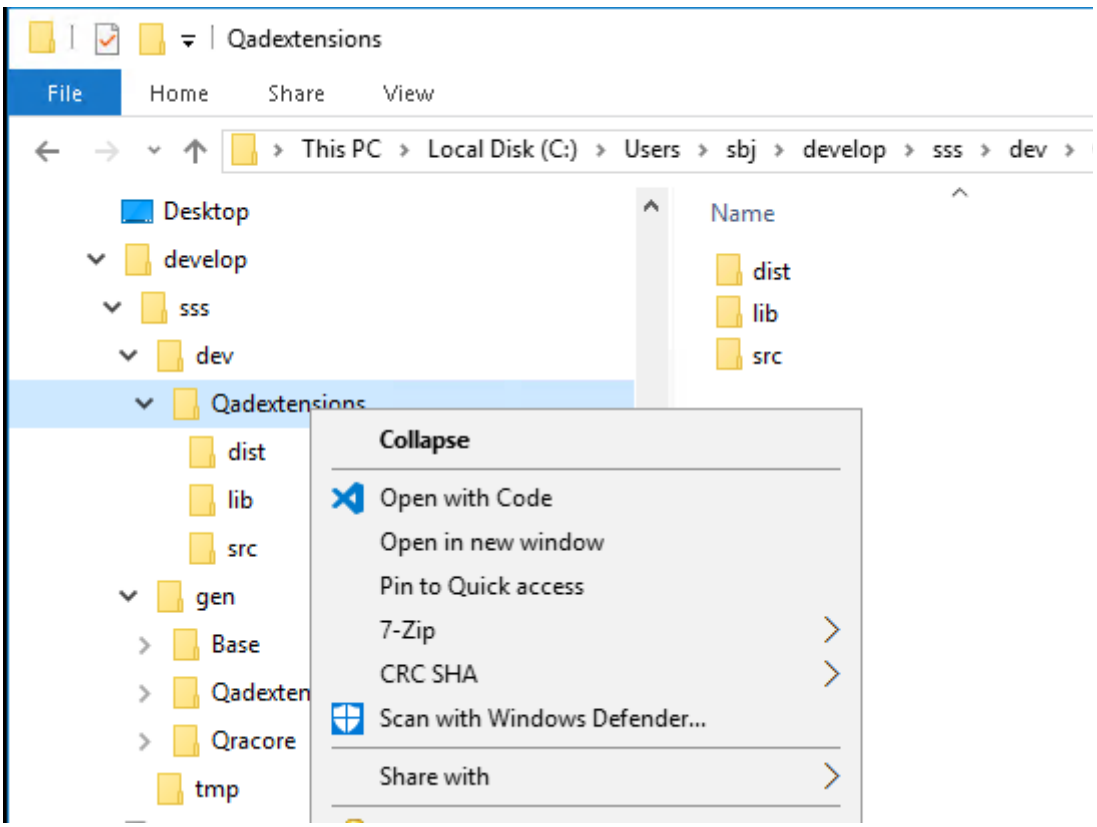
The gen folder contains the generated proxy projects. In every project folder, you see the same structure, namely three folders:

1. dist: this is the folder where the output of the project is copied into. After compiling your project, it contains a .js, a .map, and a .d.ts file for your project. These are the files that will be uploaded to the server when you are ready to develop your scripts.
2. lib: this is the folder that contains all the type definition files of the proxies you need to reference in your project.
3. src: this is where your scripts will go.

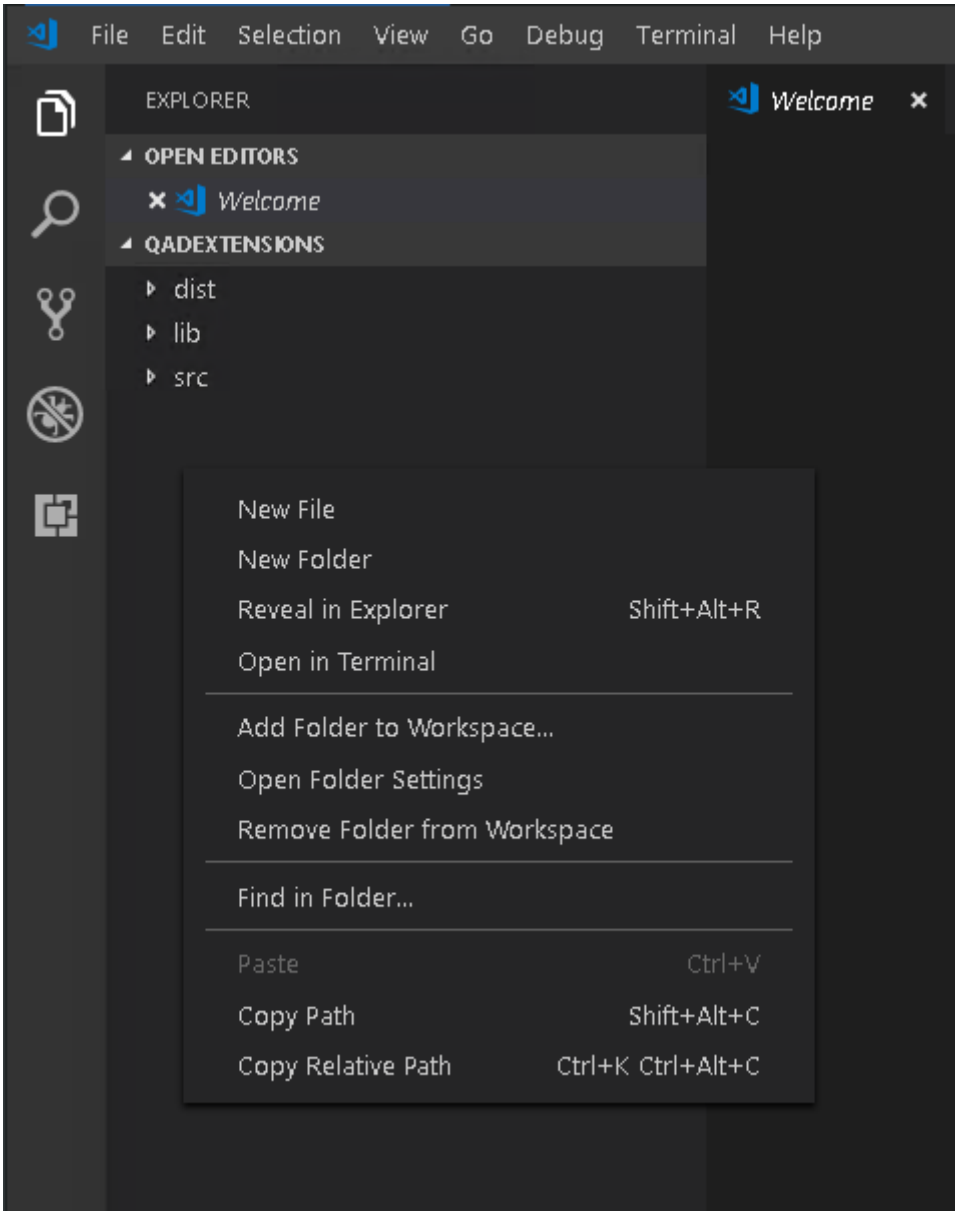
So, we'll have to create the same structure for our dev project. The only difference is that we will create it under a dev folder instead of a gen folder:



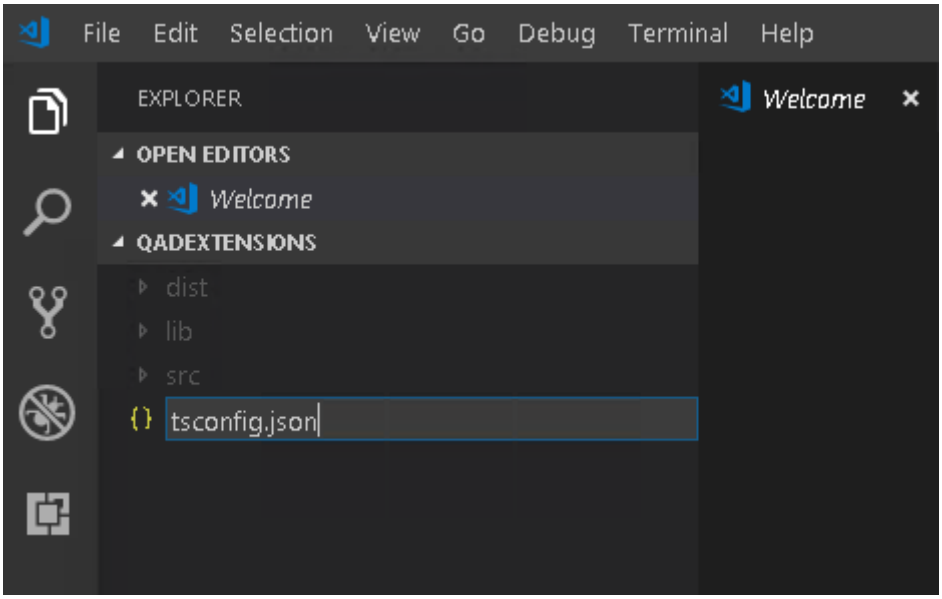
Now, the only thing missing for a TypeScript project is a tsconfig.json file. Also, this one we need to create manually in the dev/Qadextensions folder. It's best to do that in VS Code. So next, right-click the Qadextensions folder, and select **Open with Code**:



Then in code, add a new file by right-clicking the empty space underneath the project folders:



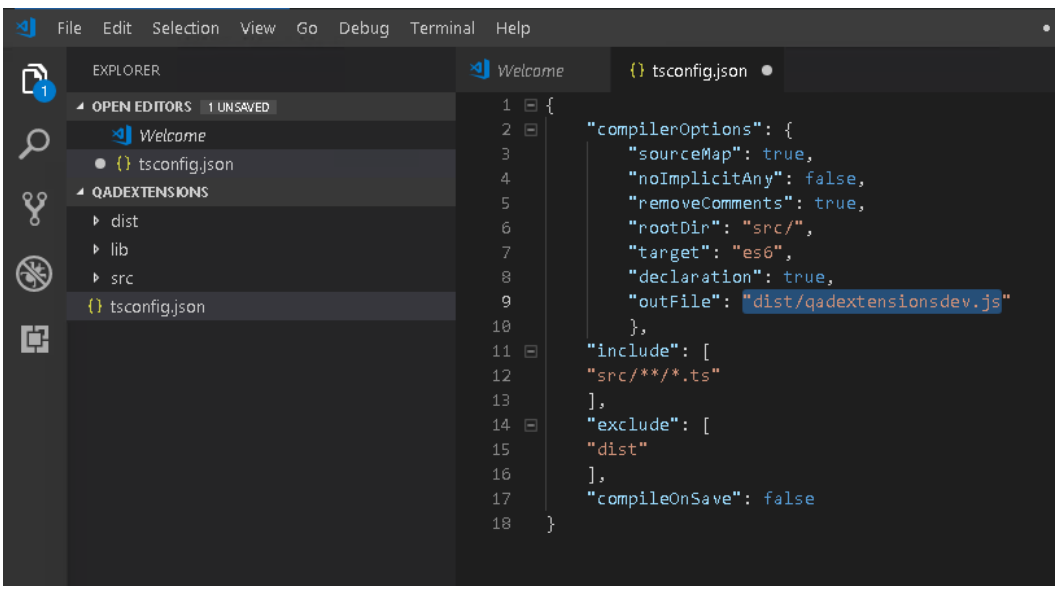
Then, type tsconfig.json:



Press Enter. It will open the tsconfig.json file. Paste the following text in it:

```
{
    "compilerOptions": {
        "sourceMap": true,
        "noImplicitAny": false,
        "removeComments": true,
        "rootDir": "src/",
        "target": "es6",
        "declaration": true,
        "outFile": "dist/qadextensionsdev.js"
        },
    "include": [
    "src/**/*.ts"
    ],
    "exclude": [
    "dist"
    ],
    "compileOnSave": false
}
```

Then, adapt the outFile name to your needs. In this example, it is already correct:



Now, save the file by pressing ctrl-s.

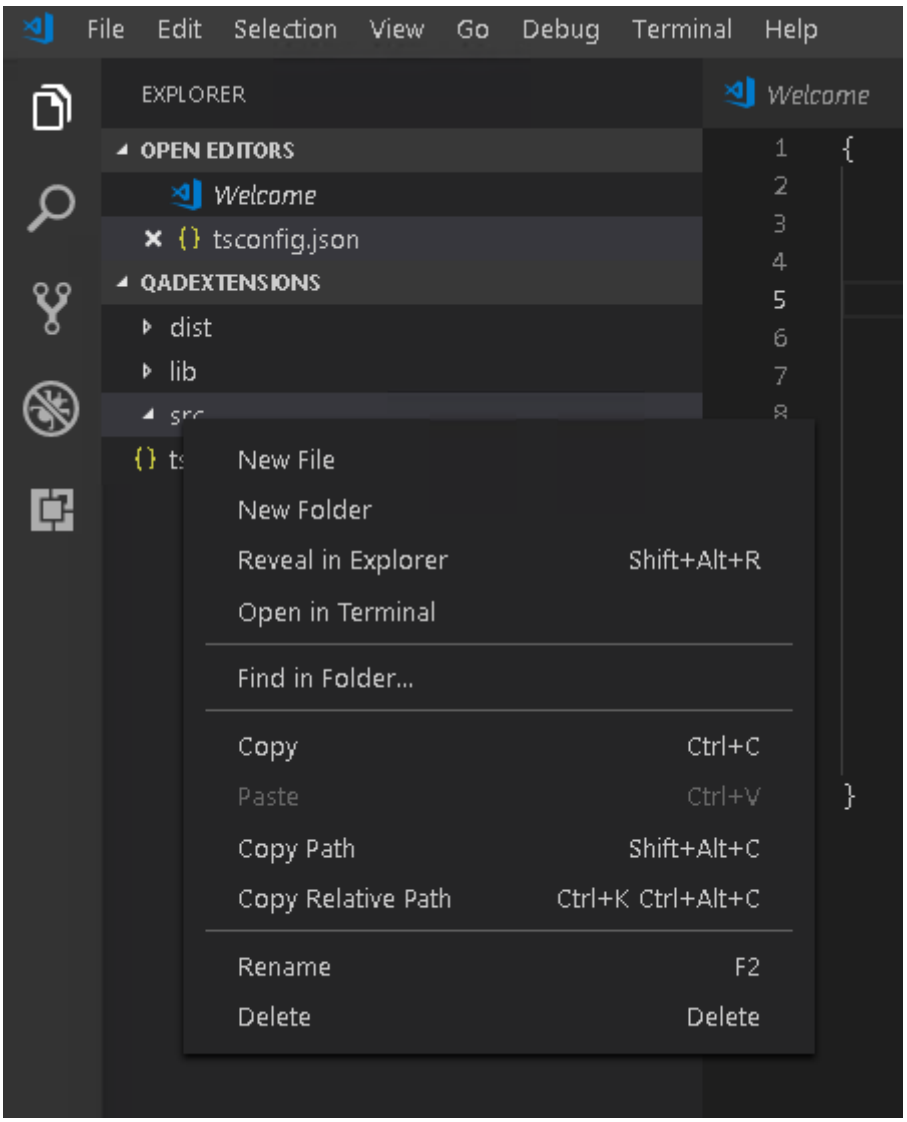
Important is that in ***tsconfig.json*** which is the configuration file for your project, you define the name for your output javascript file. This name needs to be the same as in your dev build script (the script build-qadextensions-dev, in this example).

A video that shows the above steps:

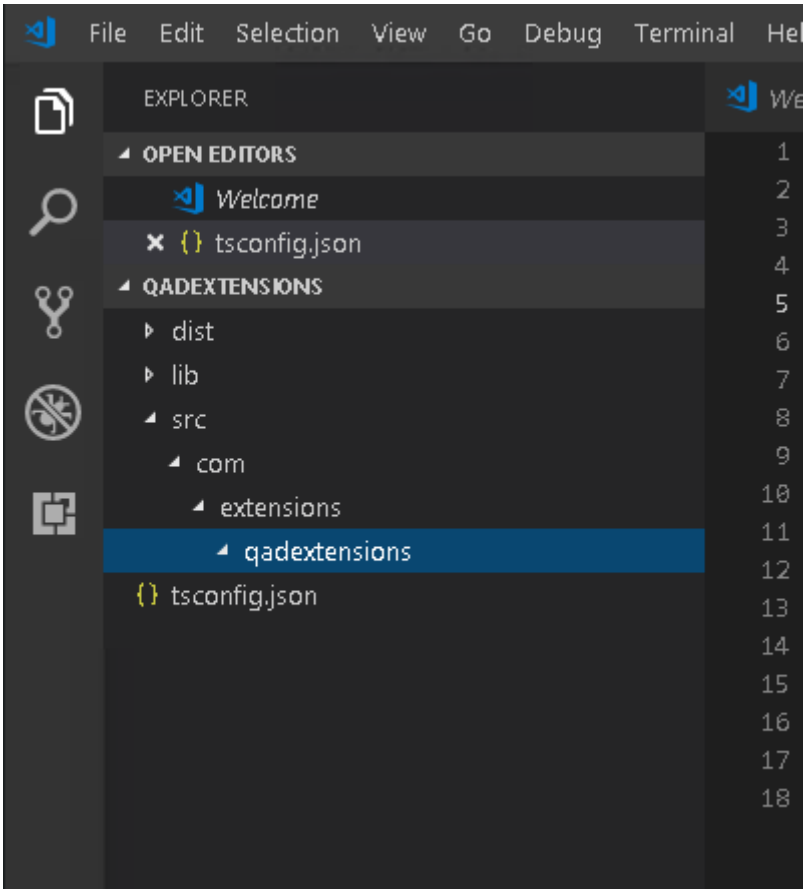
Your browser does not support the HTML5 video element

Step 2: Create src folder structure

The next thing we do is creating the folder structure for your script sources. We use the same namespace as the generated proxy of our project, namely com/extensions/qadextensions. We can do this by right-clicking the src folder, and selecting **New Folder**:

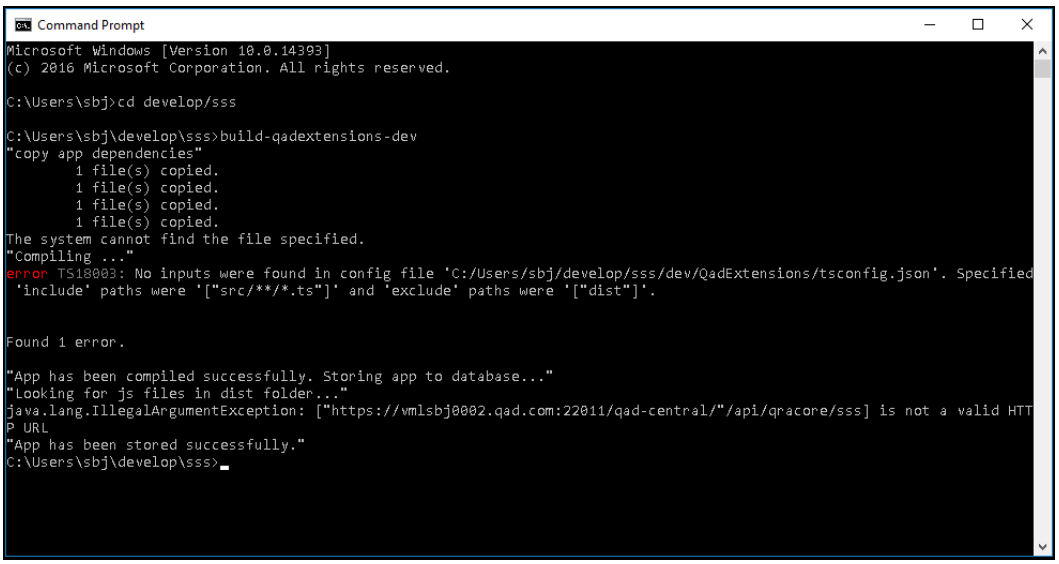


You should end up with a structure like this:

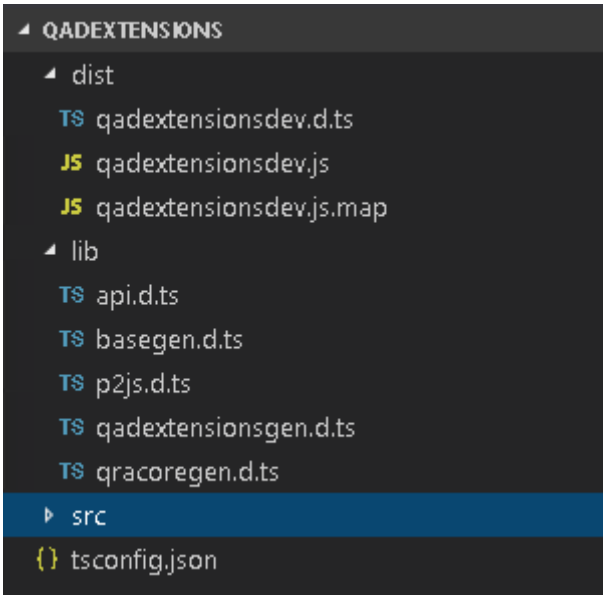


Step 3: Copy type definition files to the lib folder

As you can see, the lib folder in your project is empty. Type definition files for the apps you want to reference need to be copied in there. You can do that manually, or you can run the build-qadextensions-dev script:



Note the copies at the beginning of the output. These are the type definition files being copied to the lib folder. The rest of the script fails at this point. That's because we have no script sources yet, there is nothing to compile. In your VS Code you will, however, see the type definition files in the lib folder:



A video that shows the above steps:

Your browser does not support the HTML5 video element

After this, you are ready to start developing a script: Developing and deploying server scripts.

Developing and deploying server scripts

Introduction

These pages describe how to develop server scripts:

1. How to develop a server script: develop a server script by following an example.
2. How to deploy a server script: deploy the script developed in step 1 and test 1.
3. An explanation on the different things you can do with the server TS api.
4. An explanation on how to work with extension data in your scripts.
5. An explanation on how to work with related data in your scripts.
6. Debugging server scripts.

Detailed explanation:

Developing server scripts

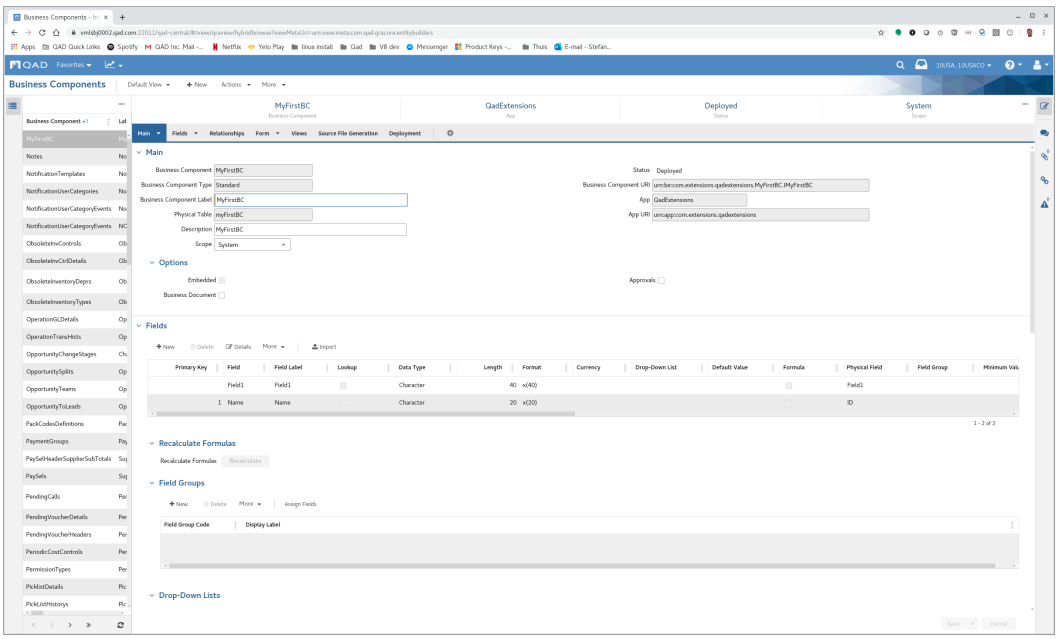
Introduction

After creating the project structure, we can start developing our scripts. This page explains how to do that with a simple example: we will create a validation on this the BC 'MyFirstBC' so that the value "CREATE" is not allowed for Field1 when creating a record, and the value "UPDATE" is not allowed when updating the record. We will create a new ts file, overwrite and implement the create and update methods, and compile at the end.

Step 1: Choose a BC to extend

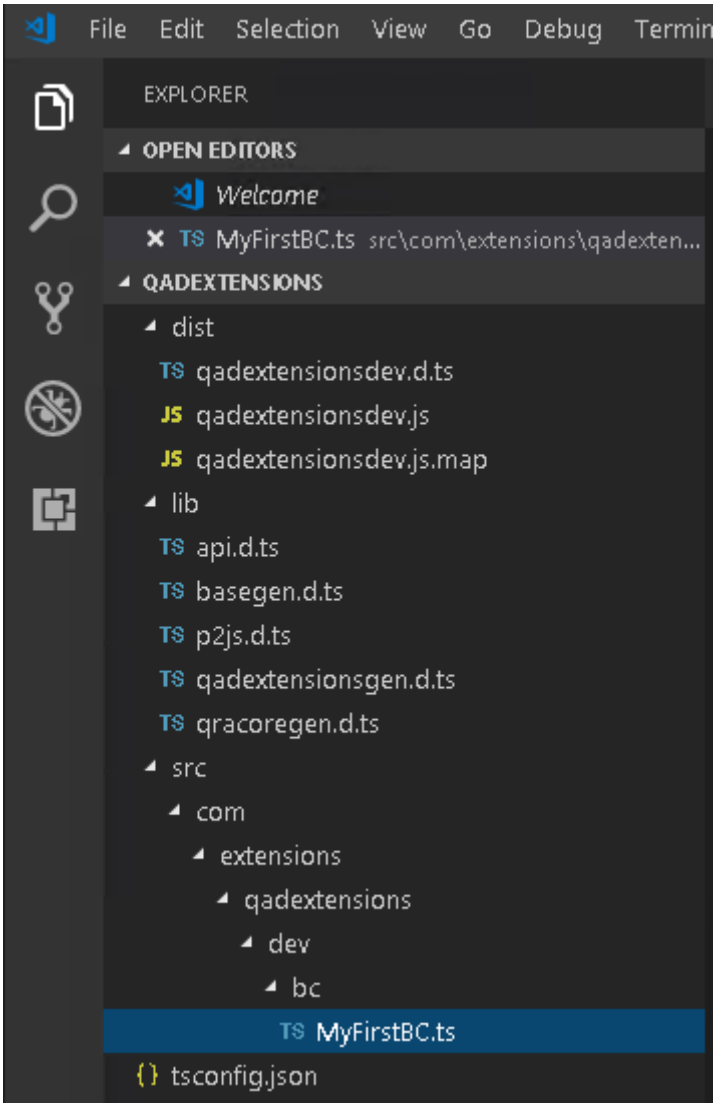
Before you can extend a Business Component with a script, you need to have a proxy for that BC .
See Setup-4: Generating, building, and deploying TypeScript proxies (script: build_<AppName>)).

In this example, we use a BC named 'MyFirstBC' which is created in our own app:



Step 2: Create the ts file for your script

The next step is to add a new ts file to your TypeScript project. You are free to create a directory structure for that where you place that file in. However, it is recommended to use a structure similar to the proxies for your project and to distinguish from it with a dev folder. We chose the filename of the ts file to be the same as the BC name:



Step 3: Add type definition references and a namespace

The next step is to add references to the type definition files and add the namespace in the file you created in the previous step:

```
/// <reference path="../../../../../../lib/api.d.ts" />
/// <reference path="../../../../../../lib/p2js.d.ts" />
/// <reference path="../../../../../../lib/qadextensionsgen.d.ts" />

namespace com.extensions.qadextensions.dev.bc {

}
```

As you can see, there are three references to .d.ts files. The first two are the TS server api, then the 3d is the proxies of the QadExtensions app.

The namespace is the same as the folder structure.

Step 4: Add a class declaration for your BC script

Now, declare the class that will extend the BC and add a constructor:

```
namespace com.extensions.qadextensions.dev.bc {
```

```
    export class MyFirstBC extends com.extensions.qadextensions.MyFirstBC.gen.bc.MyFirstBC {
        constructor(){
            super ("com.extensions.qadextensions.dev.bc.MyFirstBC");
        }
    }
}
```

The constructor calls its super constructor to pass its class name. This is for logging purposes. If you log something from within this class, this name will be prefixed to the log line.

Step 5: Add a Factory class declaration for your BC script

The server framework works with a factory that is needed to create an instance:

```
    export class MyFirstBCFactory extends com.extensions.qadextensions.MyFirstBC.gen.bc.
MyFirstBCFactory {
        public getInstance(): MyFirstBC{
            //singleton: create new instance every time it is needed.
            return new MyFirstBC();
        }
    }
```

Step 6: Add code to register your script to the service locator and the script registry

The server scripting works with a service locator just like the progress BL logic does. So, you need to add this BC script to that service locator. You also need to make the server scripting engine on the progress site aware of which methods your script will implement. That way it will know for which methods it needs to call the script, and it will not call the script for methods that are not implemented, which is better for performance.

Add this line at the top of the file, right underneath the namespace declaration:

```
 import ServiceLocator = com.qad.tsfoundation.service.ServiceLocator;
```

and add this code at the bottom, right above the last closing bracket:

```
    ServiceLocator.STATIC_INSTANCE.addService(MyFirstBC.ENTITY_URI, new MyFirstBCFactory());
    com.qad.p2js.bcscriptrunner.ScriptsRegistry.registerBCScript(
        com.extensions.qadextensions.MyFirstBC.gen.bc.MyFirstBC.ENTITY_URI,
        ["create","update"]);
```

Note the create and update method names are passed to the registry. This means that our script will be called for these two methods.

At this point, your source code should look as follows:

```
/// <reference path="../../../../../../lib/api.d.ts" />
/// <reference path="../../../../../../lib/p2js.d.ts" />
/// <reference path="../../../../../../lib/qadextensionsgen.d.ts" />

namespace com.extensions.qadextensions.dev.bc {
    import ServiceLocator = com.qad.tsfoundation.service.ServiceLocator;

    export class MyFirstBC extends com.extensions.qadextensions.MyFirstBC.gen.bc.MyFirstBC {
        constructor(){
            super ("com.extensions.qadextensions.dev.bc.MyFirstBC");
        }

    }

    export class MyFirstBCFactory extends com.extensions.qadextensions.MyFirstBC.gen.bc.
MyFirstBCFactory {
        public getInstance(): MyFirstBC{
            //singleton: create new instance every time it is needed.
            return new MyFirstBC();
```

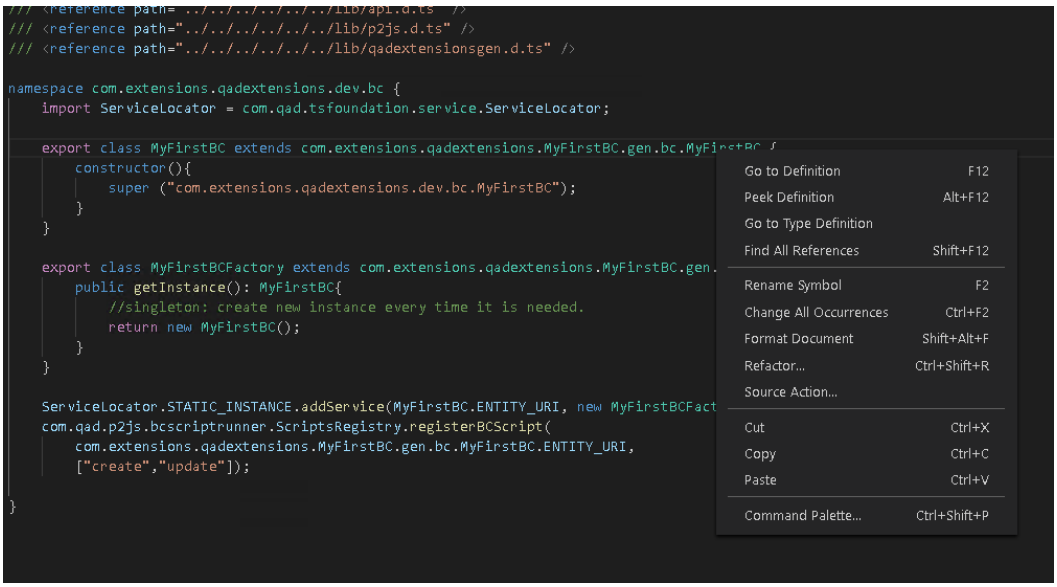
```
        }
    }

    ServiceLocator.STATIC_INSTANCE.addService(MyFirstBC.ENTITY_URI, new MyFirstBCFactory());
    com.qad.p2js.bcscriptrunner.ScriptsRegistry.registerBCScript(
        com.extensions.qadextensions.MyFirstBC.gen.bc.MyFirstBC.ENTITY_URI,
        ["create","update"]);

}
```

Step 7: Override create and update method

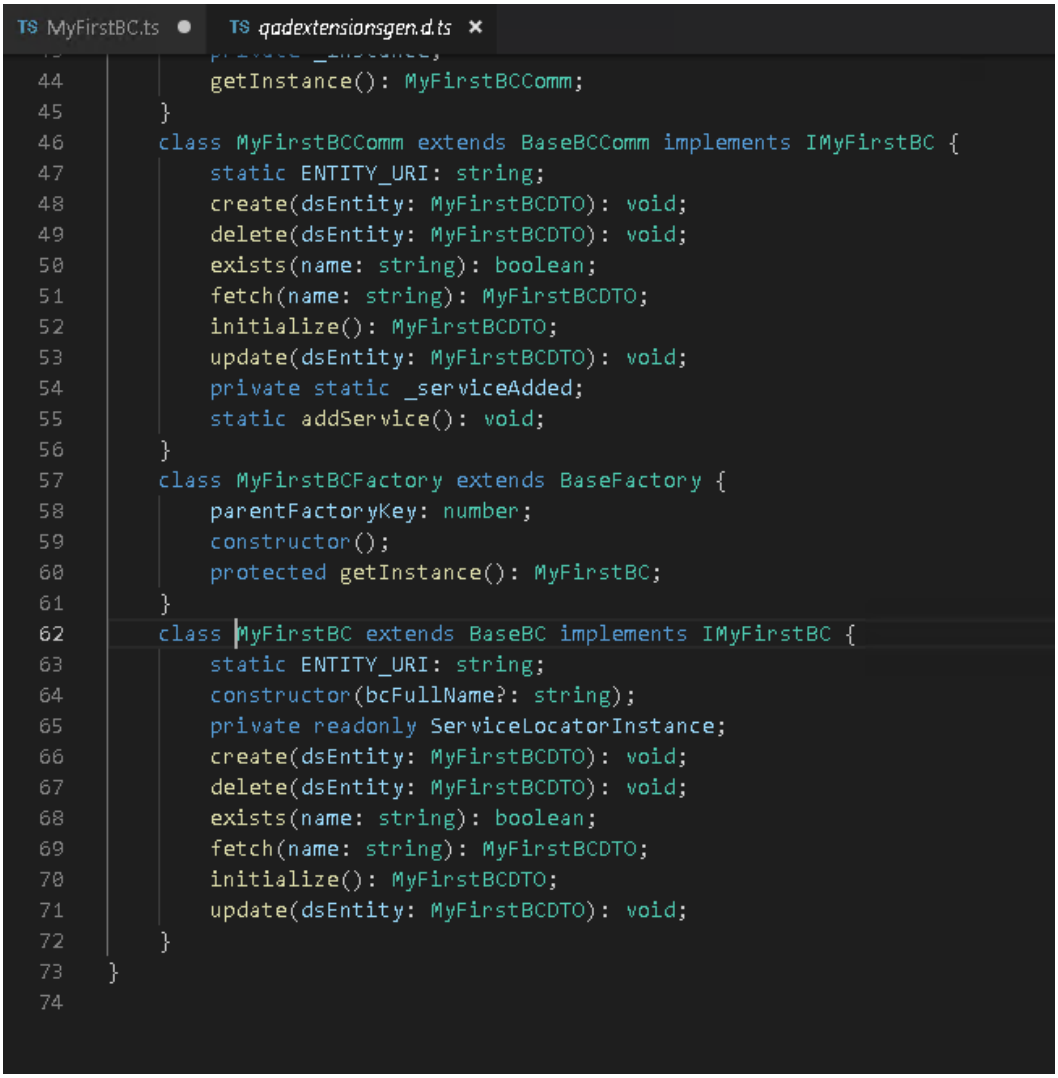
In order for our script to do something, we have to override methods from the base class. In this case, we want to override create and update.

You can simply type the method, but it's much easier to copy it from the base class. You can do that as follows:

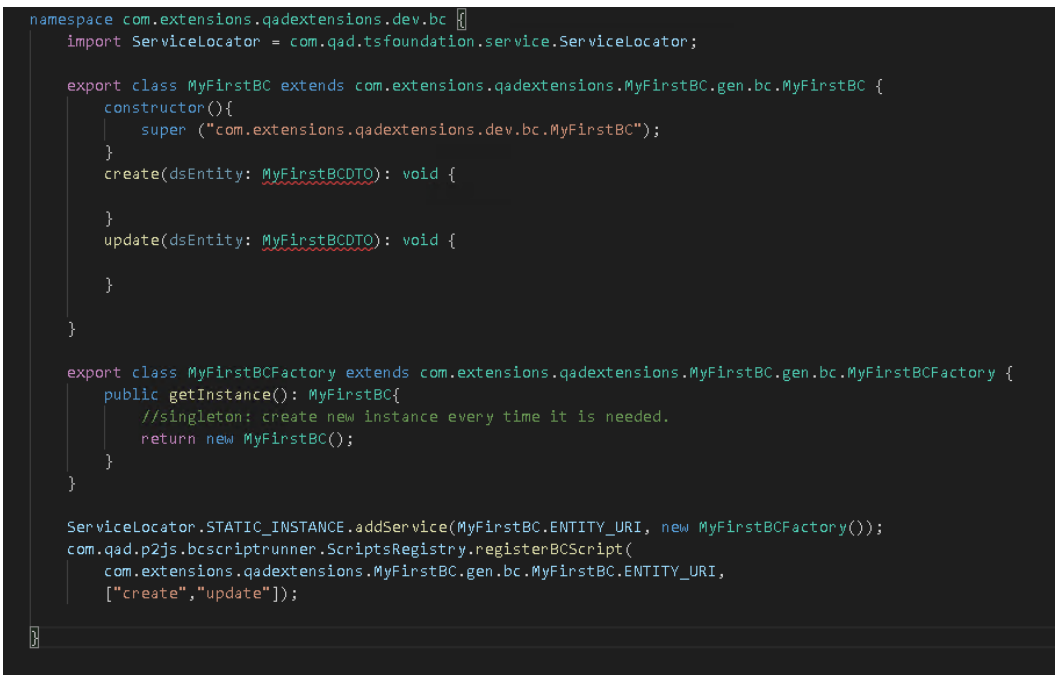
1) Right-click your class definitions at the right, on the class you extend from:



2) Click Go to Definition. It will open the super class's definition:



You can now simply copy the method declaration and add a method body:



Note the error on both function declarations. This is because MyFirstBCDTO is unknown to this script. To fix that, add this line at the top right under the other import:

```
import MyFirstBCDTO = com.extensions.qadextensions.MyFirstBC.gen.dto.myFirstBCData;
```

The class imported above is the DTO class that represents the data of the Business Component.

At this point, this should be your code:

```
/// <reference path="../../../../../../lib/api.d.ts" />
/// <reference path="../../../../../../lib/p2js.d.ts" />
/// <reference path="../../../../../../lib/qadextensionsgen.d.ts" />

namespace com.extensions.qadextensions.dev.bc {
    import ServiceLocator = com.qad.tsfoundation.service.ServiceLocator;
    import MyFirstBCDTO = com.extensions.qadextensions.MyFirstBC.gen.dto.myFirstBCData;

    export class MyFirstBC extends com.extensions.qadextensions.MyFirstBC.gen.bc.MyFirstBC {
        constructor(){
            super ("com.extensions.qadextensions.dev.bc.MyFirstBC");
        }
        create(dsEntity: MyFirstBCDTO): void {

        }
        update(dsEntity: MyFirstBCDTO): void {

        }

    }

    export class MyFirstBCFactory extends com.extensions.qadextensions.MyFirstBC.gen.bc.
MyFirstBCFactory {
        public getInstance(): MyFirstBC{
            //singleton: create new instance every time it is needed.
            return new MyFirstBC();
        }
    }

    ServiceLocator.STATIC_INSTANCE.addService(MyFirstBC.ENTITY_URI, new MyFirstBCFactory());
    com.qad.p2js.bcscriptrunner.ScriptsRegistry.registerBCScript(
        com.extensions.qadextensions.MyFirstBC.gen.bc.MyFirstBC.ENTITY_URI,
        ["create","update"]);

}
```

Step 8: Implement create and update method

The last code to add is code that will check Field1 in the data for a certain value, and throw a validation error if necessary:

```
        create(dsEntity: MyFirstBCDTO): void {
            if (dsEntity.dsmyFirstBC.ttmyFirstBC[0].Field1=="CREATE") {
                this.addValidationError("Value 'CREATE' is not allowed for Field1 when creating a
record !");
            }
            this.throwAddedValidationErrors();
            super.create(dsEntity);
        }
        update(dsEntity: MyFirstBCDTO): void {
            if (dsEntity.dsmyFirstBC.ttmyFirstBC[0].Field1=="UPDATE") {
                this.addValidationError("Value 'UPDATE' is not allowed for Field1 when updating a
record !");
            }
            this.throwAddedValidationErrors();
            super.update(dsEntity);
        }
```

Note the calls to the super method for both methods. This is really necessary or the Progress logic for the BC will NEVER run!

TypeScript is case sensitive. This means that this line:

if (dsEntity.dsmyFirstBC.ttmyFirstBC[0].Field1=="CREATE")

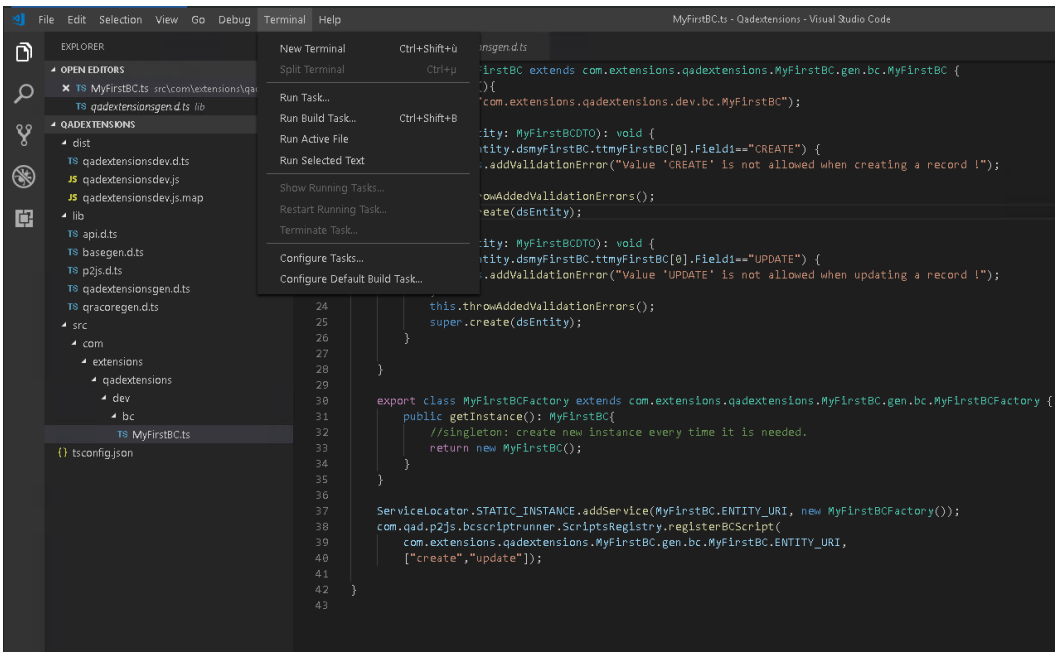
will only be true for an exact match. If you want case insensitive comparison (the default for validations), then change the line to:

if (dsEntity.dsmyFirstBC.ttmyFirstBC[0].Field1 && dsEntity.dsmyFirstBC.ttmyFirstBC[0].Field1.toUpperCase()=="CREATE")

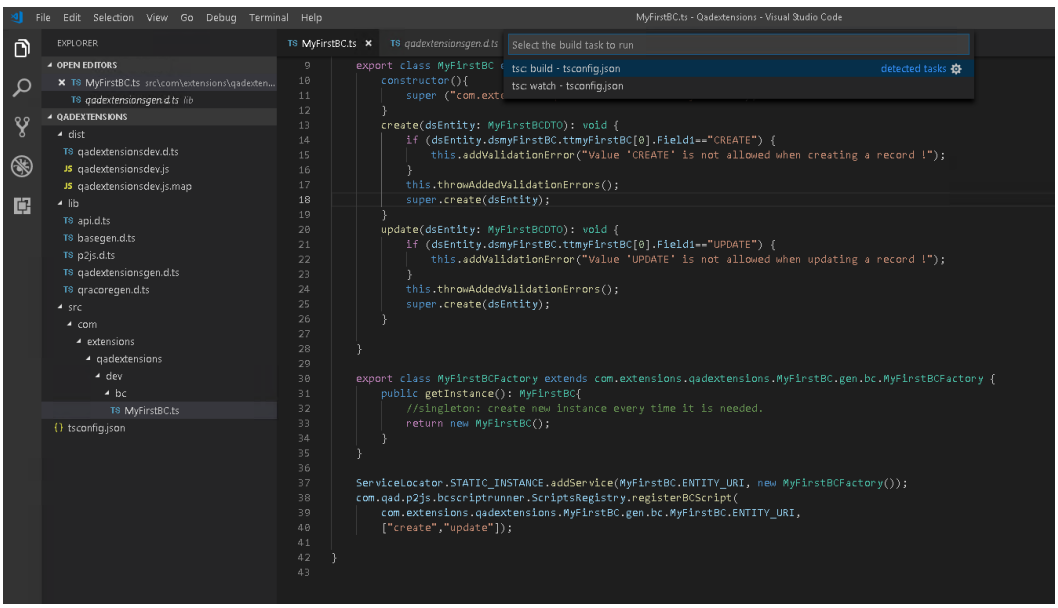
Step 9: Compile your script

The next step is to compile your script. There shouldn't be errors if you solved all the problems that VS Code shows you during development, but compiling it will, of course, make sure everything is correct. To do so:

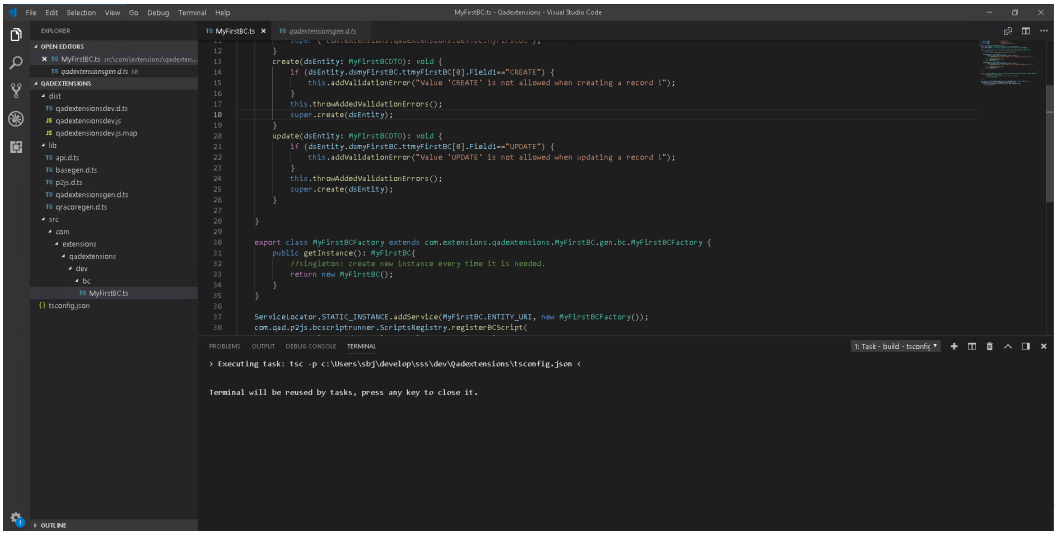
Go to Terminal and select Run Build Task:



Then, select tsc build:



That will result in a compile execution in the terminal window at the bottom of the VS Code screen:



In this example, there are no errors. If errors occur, you'll clearly see them in the terminal window.

Step 10. Next

Now, your server script is ready, the next step is to deploy it: Deploying server scripts.

Deploying server scripts

Introduction

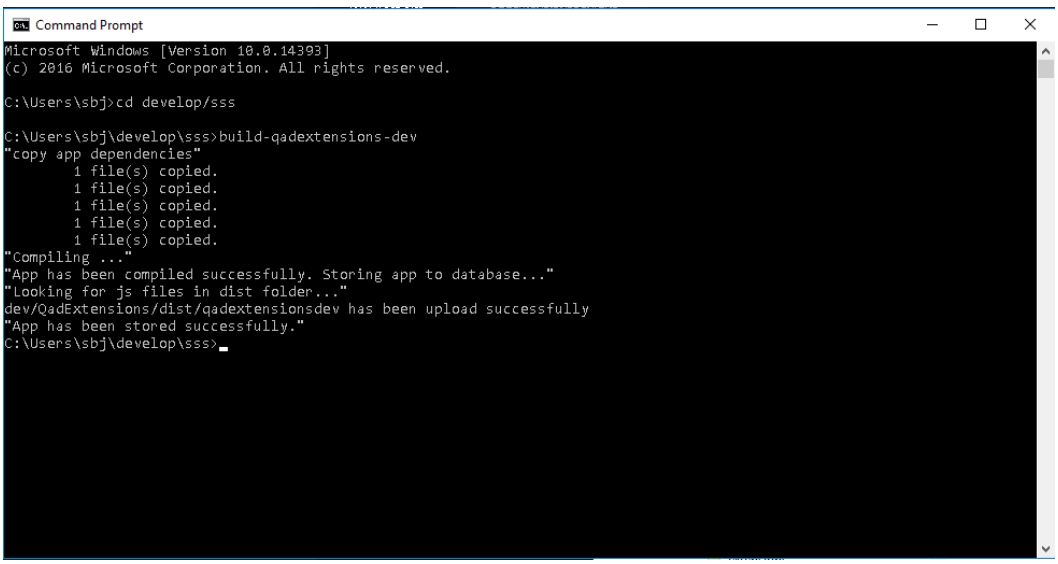
This page describes how to deploy a server-script project to the server.

Step 1: Do a test compile in Visual Studio Code

Make sure you did a test compile on your project in Visual Studio. How you do that is explained in step 9 of this page: Developing server scripts.

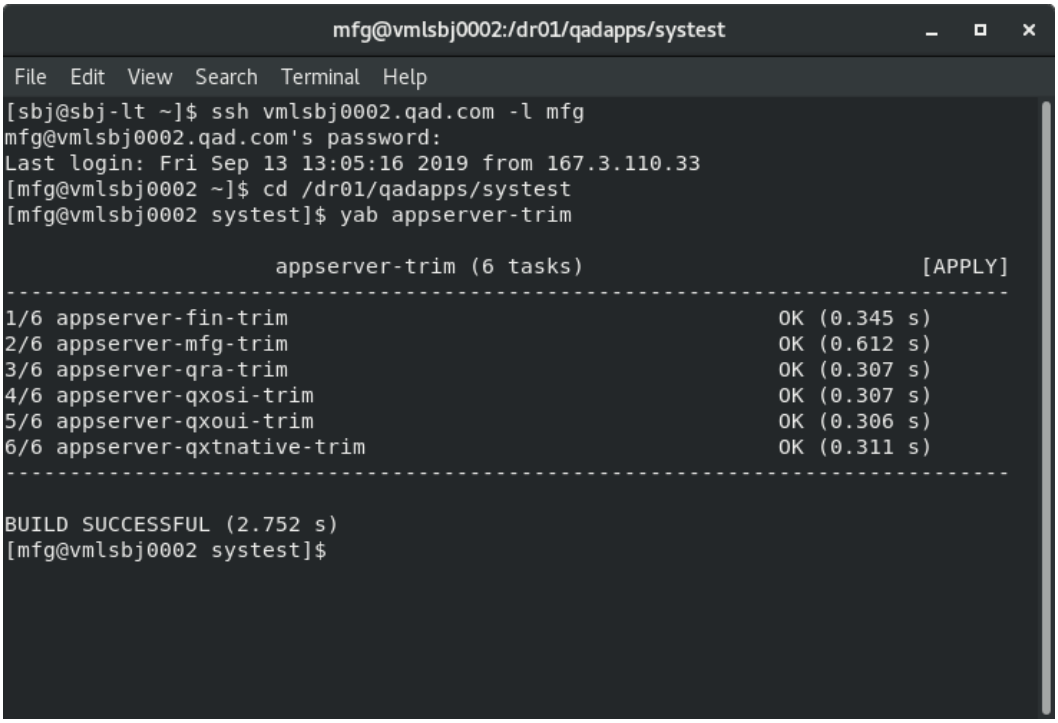
Step 2: Deploy to server (command: "build-<AppName>-dev")

In order to deploy the script to your server, you need to run the **build-qadextensions-dev** script:



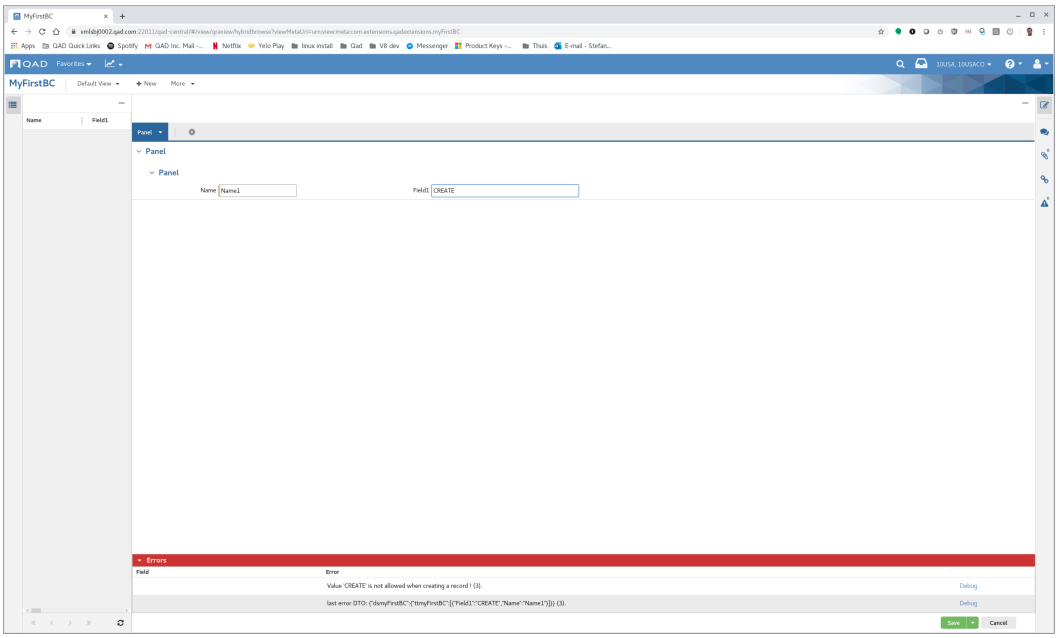
Step 3: Trim the appservers

Because server scripts are loaded on startup of an appserver during the bootstrap, you need to trim all appservers on your server by running the following command:



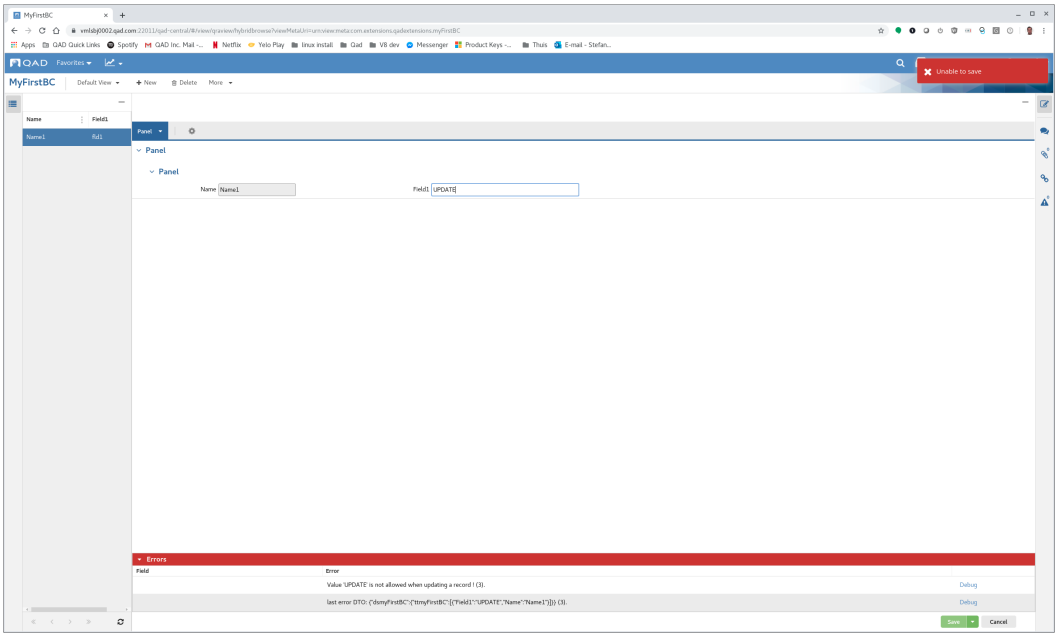
Step 4: Test your script

Open MyFirstBC from the menu, and try to create a record within Field1 value "CREATE":



Then, modify the value of Field1 to something else and save again. You'll see that it works.

Then, modify the value to UPDATE and save. It should fail again:



Modify it to something else and it should work.

Using the TypeScript API

Introduction

This page describes all the different things you can do with the server TS API.

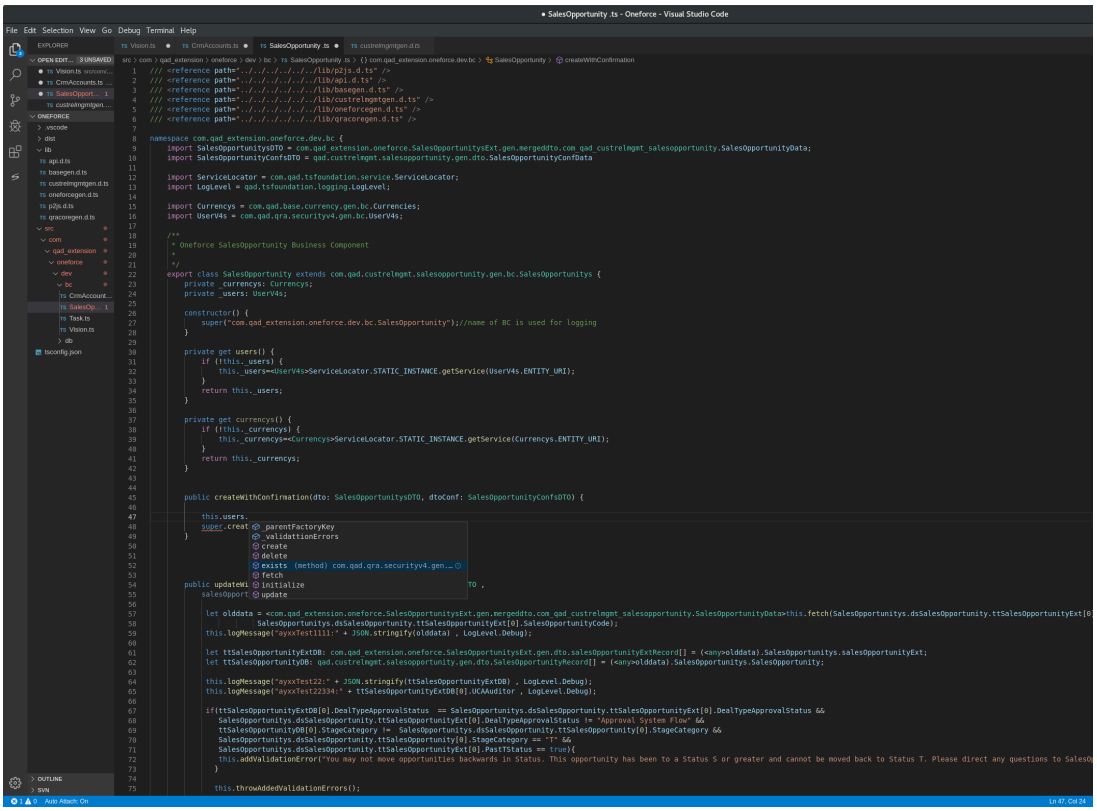
1- Calling other Business Components

In order to call out to other business components, we need to get an instance of that BC from the service locator. This can be done by calling ServiceLocator.STATIC_INSTANCE.getService, and cast it to the correct class type:

```
namespace com.extensions.oneforce.dev.bc {

    import ServiceLocator = com.qad.tsfoundation.service.ServiceLocator;
    import LogLevel = qad.tsfoundation.logging.LogLevel;

    import Currencys = com.qad.base.currency.gen.bc.Currencies;
    import UserV4s = com.qad.qra.securityv4.gen.bc.UserV4s;

    /**
     * Oneforce SalesOpportunity Business Component
     *
     */
    export class SalesOpportunitys extends com.qad.custrelmgmt.salesopportunity.gen.bc.
SalesOpportunitys {
        private _currencys: Currencys;
        private _users: UserV4s;

        constructor() {
            super("com.extensions.oneforce.dev.bc.SalesOpportunitys");//name of BC is used for
logging
        }

        private get users() {
            if (!this._users) {
                this._users=<UserV4s>ServiceLocator.STATIC_INSTANCE.getService(UserV4s.
ENTITY_URI);
            }
            return this._users;
        }

        private get currencys() {
            if (!this._currencys) {
                this._currencys=<Currencys>ServiceLocator.STATIC_INSTANCE.getService(Currencys.
ENTITY_URI);
            }
            return this._currencys;
        }
```


We are instantiating the other BCs we want to call in a property because we want to delay the instantiation of that BC to the first usage. If you would call the service locator immediately on the class variable declaration, you have a potential risk for looping when that same BC that you referenced is also referencing your BC.

After adding the above lines of code, methods can be called on the BCs:



Make sure that BCs never call each other from the constructor, otherwise they'll keep instantiating each other. For example, if users call country, and country calls back to users, both in the constructor, you'll get an endless loop!

When you are working with data of a BC that is extended by your app, you'll need to cast the dto to a merged type:

let dto=<CountriesDTO>this.countries.fetch("BE");

CountriesDTO is a merged type: com.extensions.oneforce.CountiresExt.gen.mergeddto.com_qad_base_country.CountryData;

2- Logging messages

```
            this.logMessage("We are in createWithConfirmation before the Progress code.",LogLevel.
Debug);
```

3- Adding and throwing validation errors

Adding a validation error:

```
this.addValidationError("Invalid Currency code");
```

Throwing all added validation errors:

```
this.throwAddedValidationErrors();
```

4- Working with transactions

```
    import TransactionManager = com.qad.tsfoundation.transaction.TransactionManager;

        ...

            let trnasactionID = TransactionManager.startTransaction();
            try {
                // do something

                TransactionManager.commitTransaction(trnasactionID);
            } catch (e) {
                TransactionManager.rollbackTransaction(trnasactionID);
            }
```

5- Getting translations

```
    import TranslationManager = com.qad.tsfoundation.translation.TranslationManager;
    import GetLabelsRequestRecord = com.qad.tsfoundation.translation.GetLabelsRequestRecord;

        ...

            let retStr="";
            let stringCodes:GetLabelsRequestRecord[]=[];
            stringCodes.push({stringCode: "mfg-OK", localeCode: "nl-NL"},{stringCode: "mfg-
CANCEL", localeCode: "nl-NL"})

            let translated=TranslationManager.getTranslations(stringCodes);
            for (let i=0;i<translated.length;i++){
                retStr+=translated[i].localeCode + " , " + translated[i].stringCode + " , " +
translated[i].labelTranslation + " - ";
            }

            return retStr;
```

Note that the localeCode can be undefined, in that case the current running user's locale will be used.

6- Getting the current user id and current user role ids

```
    import SessionInfo = com.qad.tsfoundation.session.SessionInfo;

        ...

        private getCurrentUserId():string {
            return "" + SessionInfo.getUserID();
        }
        private getCurrentUserRoleIDs():string {
            let retStr="";
            let roleIds=SessionInfo.getRoleIDs();
            for (let i=0;i<roleIds.length;i++) {
                if (i>0) {
                    retStr += " , "
                }
                retStr += roleIds[i];
            }
            return retStr;
        }
```

7- Working with Data Objects

A Data Object is a reusable query object. You can use it to read data from the database. First, you need to create an interface for the return data, and a class that inherits from com.qad.tsfoundation.base.db.BaseDO. In the class's constructor, you need to pass the tables, where conditions, and result fields to the super constructor:

```
/// <reference path="../../../../../../lib/p2js.d.ts" />
/// <reference path="../../../../../../lib/api.d.ts" />
/// <reference path="../../../../../../lib/basegen.d.ts" />
/// <reference path="../../../../../../lib/custrelmgmtgen.d.ts" />
/// <reference path="../../../../../../lib/oneforcegen.d.ts" />

namespace com.extensions.oneforce.dev.db {
    export interface TestDO1ResultRecord {
        Country_ID: number;
    }

    export class TestDO1 extends com.qad.tsfoundation.base.db.BaseDO<TestDO1ResultRecord> {
        constructor() {
            let tables=["Country"];
            let whereConditions=["Country.CountryIsActive=false"];
            let resultFields=["Country.Country_ID","character"]
            super(tables,whereConditions,resultFields);
        }
    }

}
```

The number of elements in the where conditions in the array needs to be the same as the number of tables. The result fields array needs to be an array with fields names / field data types pairs (field data type should always be right after field name). The field data types are Progress data types (character, integer, int64, decimal, logical, date, datetime, datetime-tz)

After that, you have a class that you can use to query:

```
import TestDO1 = com.extensions.oneforce.dev.db.TestDO1;

...

            let testDO=new TestDO1();
            testDO.run();
            let retStr="";

            for (let i=0;i<testDO.ResultDataSet.length;i++) {
                if (i>0) {
                    retStr+=",";
                }
                retStr+=testDO.ResultDataSet[i].Country_ID;
            }

            return retStr;
```

Larger example of a data object :

```
/// <reference path="../../../../../../../lib/p2js.d.ts" />
/// <reference path="../../../../../../../lib/api.d.ts" />

namespace com.extensions.manifesto.OperationActivity.dev.db {
    export interface WorkOrderResultRecord {
        wo_bom_code: string;
        wo_domain:string;
        wo_part:string;
        wr_mch:string;
        wr_op:number;
        wo_line:string;
        wr_nbr:string;
        wr_qty_comp:number;
        wo_routing:string;
        wo_site:string;
        wr_wkctr:string;
    }

    export class WorkOrderDataObject extends com.qad.tsfoundation.base.db.
BaseDO<WorkOrderResultRecord> {
        constructor(domainCode: string, productionOrder: string, orderId: string, operation:
```

```
number) {
            let tables=["wo_mstr","pt_mstr","wr_route","wc_mstr"];
            let whereConditions=[
                "wo_mstr.wo_status<>'c' and wo_mstr.wo_status<>'p' and wo_mstr.wo_domain="
+WorkOrderDataObject.convertToWhereValueString(domainCode)+" and wo_mstr.wo_lot="+
WorkOrderDataObject.convertToWhereValueString(""+orderId),
                "pt_mstr.pt_domain=wo_mstr.wo_domain and pt_mstr.pt_part=wo_mstr.wo_part",
                "wr_route.wr_domain=wo_mstr.wo_domain and wr_route.wr_lot=wo_mstr.wo_lot and
wr_route.wr_nbr="+WorkOrderDataObject.convertToWhereValueString(productionOrder)+" and wr_route.
wr_op="+WorkOrderDataObject.convertToWhereValueNumber(operation),
                "wc_mstr.wc_domain=wo_mstr.wo_domain and wc_mstr.wc_mch=wr_route.wr_mch and
wc_mstr.wc_wkctr=wr_route.wr_wkctr"
            ];
            let resultFields=[
                "wo_mstr.wo_bom_code","character",
                "wo_mstr.wo_domain","character",
                "wo_mstr.wo_part","character",
                "wr_route.wr_mch","character",
                "wr_route.wr_op","integer",
                "wo_mstr.wo_line","character",
                "wr_route.wr_nbr","character",
                "wr_route.wr_qty_comp","decimal",
                "wo_mstr.wo_routing","character",
                "wo_mstr.wo_site","character",
                "wr_route.wr_wkctr","character"
            ]
            super(tables,whereConditions,resultFields);
        }

        private static convertToWhereValueString(val:string) : string {
            if (val) {
                return "'"+val+"'";
            } else {
                return "''";
            }
        }
        private static convertToWhereValueNumber(val:number) : string {
            if (val) {
                return ""+val;
            } else {
                return "?";
            }
        }
    }

}
```

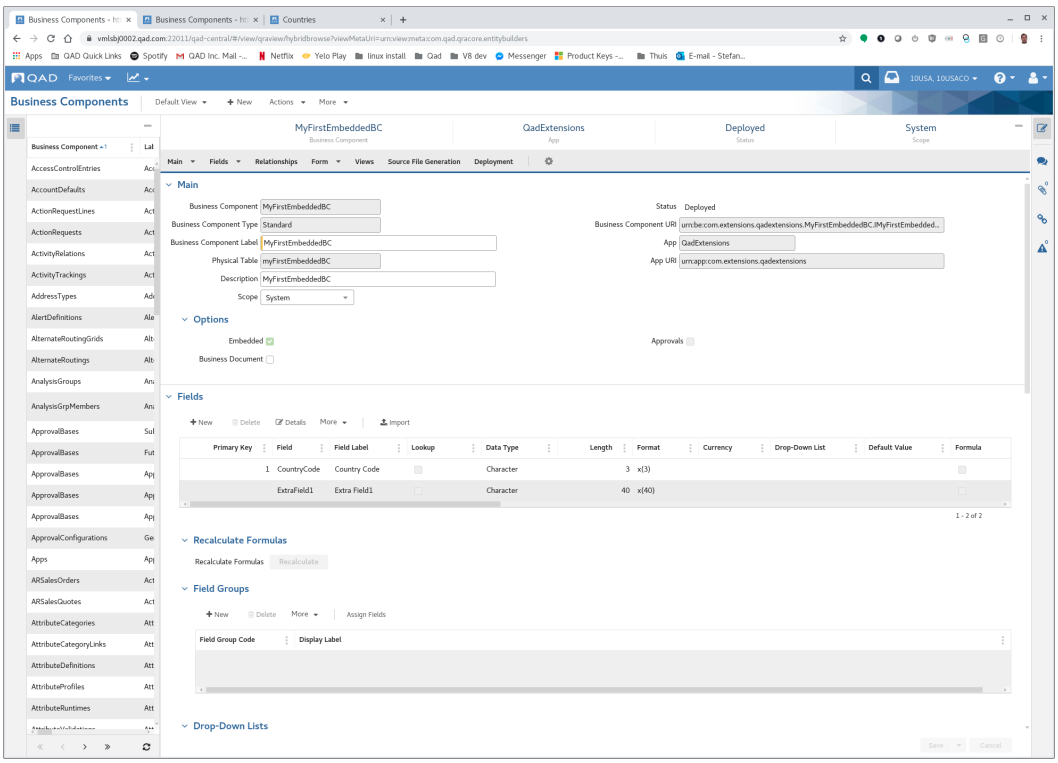
Working with extension data

Introduction

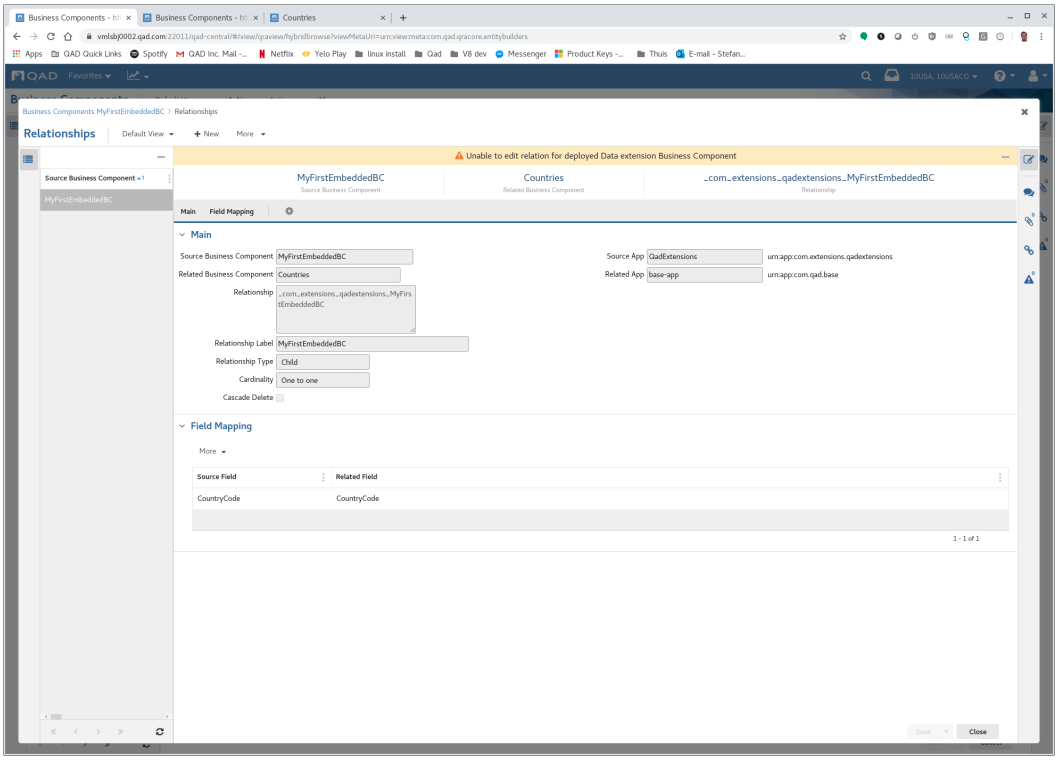
When extending BCs with an embedded BC, it is important to know how to access this embedded BC data. This page describes how to do that.

Example embedded BC

On this page, we use the following example embedded BC:



With the following relationship:



Merged DTO

In the above example, we extended Countries with an embedded BC. If we now want to do something with the embedded BC, we need to go to the BC that is extended with the embedded BC. In this case, this is Countries.

In order to see the extension fields in a script extending Countries, we need a DTO type definition that has both the Countries DTO and the embedded BC DTO. A script without embedded BC data would look as follows:

```
/// <reference path="../../../../../../lib/api.d.ts" />
/// <reference path="../../../../../../lib/p2js.d.ts" />
/// <reference path="../../../../../../lib/basegen.d.ts" />
/// <reference path="../../../../../../lib/traininggen.d.ts" />

namespace com.extensions.training.Training.dev.bc {
    import ServiceLocator = com.qad.tsfoundation.service.ServiceLocator;
    import CountriesDTO = com.qad.base.address.gen.dto.CountryData;

    export class Countries extends com.qad.base.address.gen.bc.Countries {
        constructor(){
            super ("com.extensions.training.Training.dev.bc.Countries");
        }
        create(dto: CountriesDTO): void {
            super.create(dto);
        }
        update(dto: CountriesDTO): void {
            super.update(dto);
        }

    }

    export class CountriesFactory extends com.qad.base.address.gen.bc.CountriesFactory {
        public getInstance(): Countries{
            //singleton: create new instance every time it is needed.
            return new Countries();
        }
    }

    ServiceLocator.STATIC_INSTANCE.addService(Countries.ENTITY_URI, new CountriesFactory());
    com.qad.p2js.bcscriptrunner.ScriptsRegistry.registerBCScript(
        Countries.ENTITY_URI,
        ["create","update"]);
```

```
}
```

The line that determines the DTO type is as follows:

```
import CountriesDTO = com.qad.base.address.gen.dto.CountryData;
```

This DTO only contains the Country fields. If we want to see the embedded BC fields too, we need to change that line to:

```
import CountriesDTO = com.extensions.qadextensions.MyFirstEmbeddedBC.gen.mergeddto.
com_qad_base_address.CountryData;
```

As you can see, the merged DTO is declared on the embedded BC.

Accessing the embedded BC data

Accessing the embedded BC data is as simple as using the embedded bc table. In this case, it's ttMyFirstEmbeddedBC:

```
        create(dto: CountriesDTO): void {
            if (!dto.dsCountry.ttCountry[0].CountryCode || dto.dsCountry.ttCountry[0].
CountryCode=="") {
                this.addValidationError("Country code can't be empty !");
            }
            if (!dto.dsCountry.ttMyFirstEmbeddedBC[0].ExtraField1 || dto.dsCountry.
ttMyFirstEmbeddedBC[0].ExtraField1=="") {
                this.addValidationError("Extra Field 1 can't be empty !");
            }
            this.throwAddedValidationErrors();
            super.create(dto);
        }
```

When your BC create method is called from the UI, there will always be a record for the embedded BC. However, it is possible that your BC create method is called from another BC that does not pass in the extension record. For these cases, it is best to check if the table is there and if it has a record. You can add the record if necessary:

```
            if (dto.dsCountry.ttMyFirstEmbeddedBC && dto.dsCountry.ttMyFirstEmbeddedBC.
length>0) {
                dto.dsCountry.ttMyFirstEmbeddedBC=[{
                    CountryCode:dto.dsCountry.ttCountry[0].CountryCode,
                    ExtraField1:"someDefaultValue"
                }];
            }
```

Working with related data

Introduction

This page describes how to get the related data in a BC script.

Getting related data

Getting related data by using key fields

Getting related data with the id is simply done by calling the fetch method on the corresponding business component. E.g. to get the related currency data:

```
            //get the related currency data:
            if (dto.dsSalesOpportunity.ttSalesOpportunity[0].CurrencyCode) {
                let currency=this.currencys.fetch(dto.dsSalesOpportunity.ttSalesOpportunity[0].
CurrencyCode);
                this.logMessage("currency description: "+currency.dsCurrency.ttCurrency[0].
CurrencyDescription,LogLevel.Debug);
            }
```

However, this only works if you have the complete key. You can't leave one of the key fields empty.

Getting related data with a Data Object

If you want, for example, to retrieve all sales opportunity lines for a certain sales opportunity, then you need to create a data object:

```
/// <reference path="../../../../../../lib/p2js.d.ts" />
/// <reference path="../../../../../../lib/api.d.ts" />
/// <reference path="../../../../../../lib/basegen.d.ts" />
/// <reference path="../../../../../../lib/custrelmgmtgen.d.ts" />
/// <reference path="../../../../../../lib/oneforcegen.d.ts" />

namespace com.extensions.oneforce.dev.db {
    import BaseDO = com.qad.tsfoundation.base.db.BaseDO;

    export interface SalesOpportunityLinesDOResultRecord {
        SalesOpportunityLineID: string;
        LineNumber: number;
        ListPrice: number;
    }

    export class SalesOpportunityLinesDO extends com.qad.tsfoundation.base.db.
BaseDO<SalesOpportunityLinesDOResultRecord> {
        constructor(salesOpportunityCode: string,domainCode: string) {
            let tables=["SalesOpportunityLine"];
            let whereConditions=[
                "SalesOpportunityLine.SalesOpportunityCode='"+salesOpportunityCode+"'",
                "SalesOpportunityLine.DomainCode='"+domainCode+"'"
            ];
            let resultFields=[
                "SalesOpportunityLine.SalesOpportunityLineID", BaseDO.ProgressCharacter,
                "SalesOpportunityLine.LineNumber", BaseDO.ProgressInteger,
                "SalesOpportunityLine.ListPrice", BaseDO.ProgressDecimal
            ]
            super(tables,whereConditions,resultFields);
        }
```

```
    }

}
```


The data type in the resultFields array must be a progress data type. The following data types are supported:

- character (string)
- date (Date)
- datetime (Date)
- datetime-tz (Date)
- decimal (number)
- integer (number)
- int64 (number)
- logical (boolean)

The type between brackets is the corresponding JavaScript data type.

The class you inherit from (BaseDO) contains constants you can use for the Progress data types.

And then use that data object in your code as follows:

```
    import SalesOpportunityLines = com.qad.custrelmgmt.salesopportunity.gen.bc.
SalesOpportunityLines;
    import SalesOpportunityLinesDO = com.extensions.oneforce.dev.db.SalesOpportunityLinesDO;
    import SalesOpportunityLinesDOResultRecord = com.extensions.oneforce.dev.db.
SalesOpportunityLinesDOResultRecord;

...


        private getSOPLinesTotalListPrice(dto: qad.custrelmgmt.salesopportunity.gen.dto.
SalesOpportunityData):number {
            let totalListPrice=0;
            let soplinesDO=new SalesOpportunityLinesDO(dto.dsSalesOpportunity.ttSalesOpportunity
[0].SalesOpportunityCode,
                dto.dsSalesOpportunity.ttSalesOpportunity[0].DomainCode);
            soplinesDO.run();

            for (let i=0;i<soplinesDO.ResultDataSet.length;i++) {
                if (soplinesDO.ResultDataSet[i].ListPrice) {
                    totalListPrice+=soplinesDO.ResultDataSet[i].ListPrice;
                }
            }

            return totalListPrice;
        }
```

Debugging server scripts

Introduction

Currently, the only way to debug server scripts is by logging messages. This page describes how to do that.

Log a message

Logging a message is done as follows:

```
import LogLevel = qad.tsfoundation.logging.LogLevel;


this.logMessage("My log message.",LogLevel.Debug);
```

Note that the log level can be set to the same values as in the Progress BL:

```
enum LogLevel {
    Error = 1,
    Warning = 2,
    Info = 3,
    Debug = 4
}
```

Log the BC data

In order to debug, you often need to log the BC data. This can be done by first serializing the data to a JSON string, and then adding it to the log message:

```
this.logMessage("BC data : \n" + JSON.stringify(dto) ,LogLevel.Debug);
```

Script identifier

Everything that you log in your BC script will be prefixed with the identifier you pass in the constructor. It's recommended to pass the full class name, to make it a unique string:

```
    export class Vision extends com.qad_extension.oneforce.Vision.gen.bc.Vision{

        constructor(){
                super ("com.qad_extension.oneforce.dev.bc.Vision");
        }
```

Training Exercise examples

Case 1: "Purchase Orders - Contract Field"

Note: When entering Purchase Orders, Buyers are required to enter the "Contract type" for the purchase when creating the PO manually. Contract Types are created and managed in Generalized Codes.

Expected result after completing the exercise:

1. Go to the “Purchase Orders” Business Component.
2. Click “New” to create the new invoice record.
3. Under “Billing”, see the “Contract” field with the lookup to the Generalized Codes that will pre-filter needed code type.
4. Validate on save that the value exists and it is a valid value for the generalized code; if the validation fails, we display a warning.
5. Make the "Contract" field mandatory field when creating the PO.

Hint: Use the existing “Contract” field on PO, enable Lookup to Generalized Codes by going to the lookup definition. Validate on save that the value exists, and it is a valid generalized code value by adding Server-Side Scripting.

```
// we create an instance of Generalized code bc to check if such code exists
private generalizedCodes: GeneralizedCodes = <GeneralizedCodes>ServiceLocator.STATIC_INSTANCE.
getService(GeneralizedCodes.ENTITY_URI);
    // as we perform check while creating bc, we should override create method
    public create(dto: com.qad.purchasing.purchaseorders.gen.dto.PurchaseOrderData): void {
      const record = dto.dsPurchaseOrder.ttPurchaseOrder[0];
      // we check that po_contract field with corresponding value exists in generalized codes
      if (!this.generalizedCodes.fetch(record.DomainCode, "po_contract", record.Contract)) {
        // if not - we add an error
        this.addValidationError("missed contract value");
      }
          this.throwAddedValidationErrors();
      // to be sure that our event will be triggered, we should call super
      super.create(dto);
  }
```

How to deploy script to the server

Case 2: "Supplier/Customer Invoices with Posting Date = Invoice Date"

Note: When creating a new Customer Invoice or Supplier Invoice, you are expected to add validation where the Posting Date will always equal Invoice Date. Use Server-Side and Client-Side Scripting capabilities.

Exercise #1. Expected result after the exercise is complete:

1. Go to the “Customer Invoices” Business Component.
2. Click “New” to create the new invoice record.
3. Choose “Invoice Type”.
4. Enter “Invoice Date”.
5. On leaving the field, see that the “Posting Date” automatically populates with the same date as the “Invoice Date” field if inactive.

Hint:

```
export class CustomerInvoicesFormHandler extends QraViewFormTSHandlerV2<DTO.
CustomerInvoicesMaint> {
          // this event will be triggered when we leave any field in the form
      public onFieldLeave(viewField: IViewField<any>, eventData: EventData.QraView.
FieldLeaveEventData<any>) {
          const IsInvoice: boolean = this.ViewController.getViewField("CustInvoiceTypeAutoField").
Value == "INVOICE";
                    // now we can check what field exactly has been called by checking 'fieldName'
property of the eventData parameter
          if (eventData.fieldName == "CustInvoiceDateAutoField" && IsInvoice) {
                          // and when we ensured that this is field we need we can perform some
logic
                          const PostingDate = this.ViewController.getViewField
("CustInvoicePostingDateAutoField");
                PostingDate.Value = eventData.fieldValue;
                          PostingDate.IsDisabled = true;
```

```
        }
    }
}
```

Exercise #2. Expected result after the exercise is complete:

1. Go to the “Customer Invoices” Business Component.
2. Click “New” to create the new invoice record.
3. Choose “Invoice Type”.
4. Enter “Invoice Date”.
5. Enter “Posting Date” different from the “Invoice Date”.
6. Enter all required data for the new Invoice.
7. Click Save.
8. Display a warning flash message to a user so they know they need to adjust the “Posting Date” to match the “Invoice Date”.

***Hint**:*

```
// as we perform check while creating bc, we should override create method
public create(dto: com.qad.financials.salesledger.customerinvoice.gen.dto.CustInvoiceData): void {
      const record = dto.dsCustInvoice.ttCustInvoice[0];
      if (record.CustInvoicePostingDate != record.CustInvoiceDate) {
        this.addValidationError("Please adjust a Posting date.");
      }
         // and show errors on the ui
      this.throwAddedValidationErrors();
      // don't forget to call super as otherwise it won't get called
      super.create(dto);
    }
```

Case 3: "Prevent Sales Order Confirm when Order On Hold (with the status HD)"

Note: Set “Confirmed” flags on both SO Header and Sales Order Lines to have a default value = unchecked.

Exercise #1. ***Expected result after the exercise is complete:***

1. Go to the “Sales Orders” Business Component.
2. Click “New” to create the new order record.
3. See that the default value for the "Confirmed" checkbox **= NO by default**

Hint: You can add the default value through the design layout. (The alternative way of completing the exercise is to use Server-Side Scripting).

```
public initialize(dto: com.qad.sales.salesorder.gen.dto.SalesOrderHeaderData): void {
      const record = dto.dsSalesOrderHeader.ttSalesOrderHeader[0];
      // default unchecking case
      record.IsConfirmed = false;
      super.initialize(dto);
    }
```

Exercise #2. ***Expected result after the exercise is complete:*** Handle the Action Status on “HD”. If the Action Status set to “HD”, the system to uncheck the "Confirmed" checkbox and flash the warning message "Order cannot be confirmed with an HD action status. Sales Order confirmed field was unchecked."

1. Create new SO.
2. Choose Actions Status = HD.
3. Exit the AS field, notice that the system automatically unchecked the "Confirmed" checkbox.
4. See a warning message with the following text: "Order cannot be confirmed with an "HD" action status. Sales Order confirmed field was unchecked."
5. Additional scenario to cover: After seeing the message, the user removes the “HD” value, the system to set the "Confirmed" checkbox back to the checked status.

Hint:

```
public onFieldLeave(viewField: IViewField<any>, eventData: EventData.QraView.
FieldLeaveEventData<any>) {
```

```
        if (eventData.fieldName == "ActionStatusAutoField1" && eventData.fieldValue == "HD") {
                  // todo: show warning
                  // Order cannot be confirmed with an HD action status. Sales Order confirmed
field was unchecked.
                  this.ViewController.getViewField("IsConfirmedField").Value = false;
          }
}
```

Exercise #3. Expected result after the exercise is complete: Extend Sales Order Confirm bulk action to exclude Sales Orders that have Action Status = HD.

1. Verify no SO lines are loaded into lines grid on action when Action Status = HD.
2. Bulk Confirm > Continue Button > SO grid. Exclude SO with **Action Status = HD**
3. Verify SO can be confirmed when Action Status = HD is removed (need to do this in the Sales Order Credit Screen).
4. Check if SO available in the grid after Action Status update (<>HD) (Use **Sales Order Credit** form for the Action Status update).

Hint: Use Client-Side Scripting capabilities to complete the exercise.

```
public onFieldLeave(viewField: IViewField<any>, eventData: EventData.QraView.
FieldLeaveEventData<any>) {
        if (eventData.fieldName == "ActionStatusAutoField1" && eventData.fieldValue == "HD") {
                  this.ViewController.getViewField("IsConfirmedField").Value = false;
                // check the grid
                const grid = this.ViewController.getViewGrid("");
                let gridRowDataBeforeStatusChange = (<Qad.QraView.UI.ViewGrid><any>grid).
correctGridTimezone(grid .getSelectedRowData());
                    for (let rowData of gridRowDataBeforeStatusChange) {
                        // check if so line here
                        let firstCol = rowData[0];
                        if (firstCol == "Sales Order") {
                                // show warning
                                console.log("SO presents");
                        }
                }
                // change status
                const actionStatus = this.ViewController.getViewField("ActionStatusAutoField1");
                actionStatus.Value = "HO";
                let gridRowDataBeforeStatusChange = (<Qad.QraView.UI.ViewGrid><any>grid).
correctGridTimezone(grid .getSelectedRowData());
                    for (let rowData of gridRowDataAfterStatusChange) {
                        // check if so line here
                        let firstCol = rowData[0];
                        if (firstCol == "Sales Order") {
                                // show warning
                                console.log("SO presents");
                        }
                }
        }
}
```