brixxbox documentation

brixxbox Wiki

This is the documentation for the brixxbox Low Code Application Plattform

Visit us under: www.brixxbox.net

Open the brixxbox app here

Watch some Videos to get started

Brixxbox Low-Code Plattform

Configuration

Configuration is a folder, present of the left side-bar of Brixxbox web-app. It includes all essential options to create, update, manage and customize workspaces of a particular user. Below you can find the working summary of each options present under configuration folder. To read about each option in detail, click on "click here" at the end of each option.

Last Edited

Brixxbox allow users to create and update multiple applications. In order to increase usability, it provides last edit option. User will be able to see all the recently edited app's here and save time. For example, consider your workspace consists of 100 applications and you were working on one app, then you don't want to go through all the apps instead you can find this app directly from here.

Apps

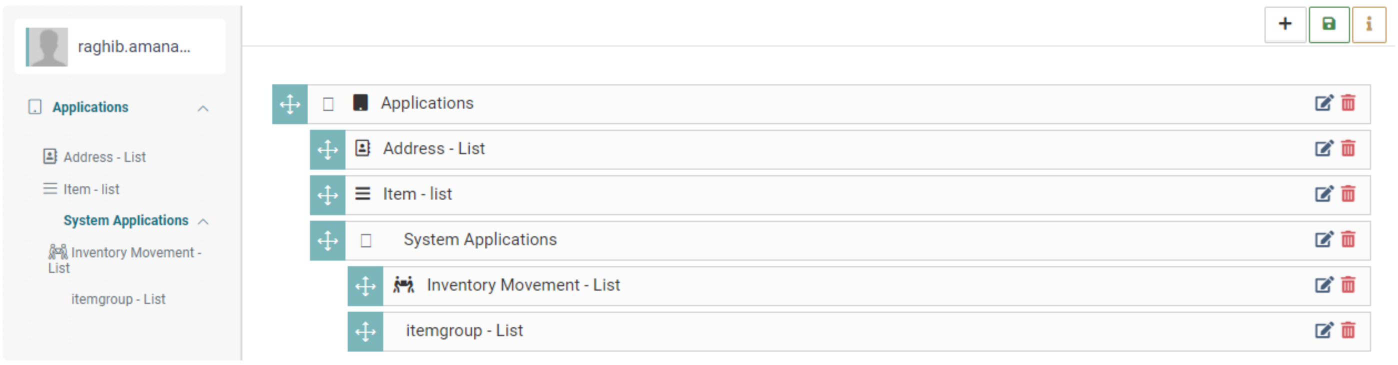
Brixxbox is all about creating user defined applications. Apps option takes user to new Brixxbox page. This page lists all the apps present in the user space. Here Brxxbox allow users to search, sort, create, edit and delete user defined applications. In order to learn more about it click here.

Template Gallery

Not only Brixxbox allows you to create your workspace from scratch, it also provides a list of templates for new users to get started with. It contains templates for beginners like "App in 4 minutes" to complete CRM systems like "Digital Address Book". In order to learn more about Brixxbox templates click here.

Menu

In Brixxbox left side panel, there are folders present like "Configuration" itself. These are all default folders. In addition to them, Brixxbox also allows user to add a customized folder. This menu option is for creating this customized folder. In this option, on first level you are only allowed to create a folder and all sub folders and apps can be placed on next hierarchical level. For example, in the menu editor we see that on first level "Applications" folder is defined an in it two apps "Address - list", "Item - list", and a folder "System Applications" are present. We can also see a same hierarchical order on left side panel.



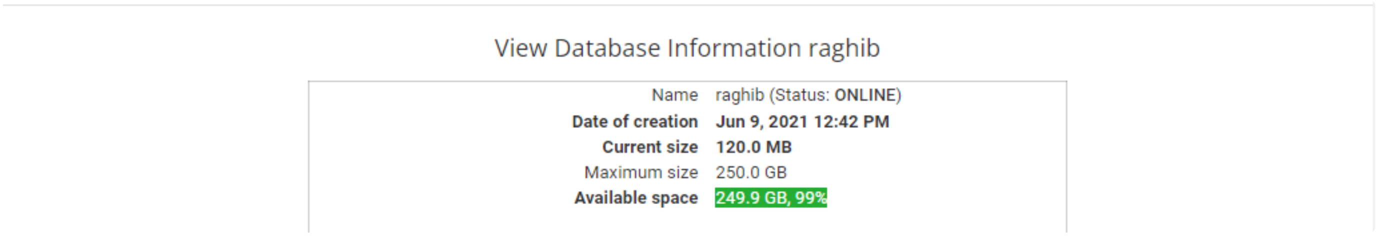
In order to learn more about it click here.

Database

In this folder, Brixxbox provides you all the functionality which is needed to create, update and maintain your database. Following are the sub options provided by Brixxbox to manage database.

Info

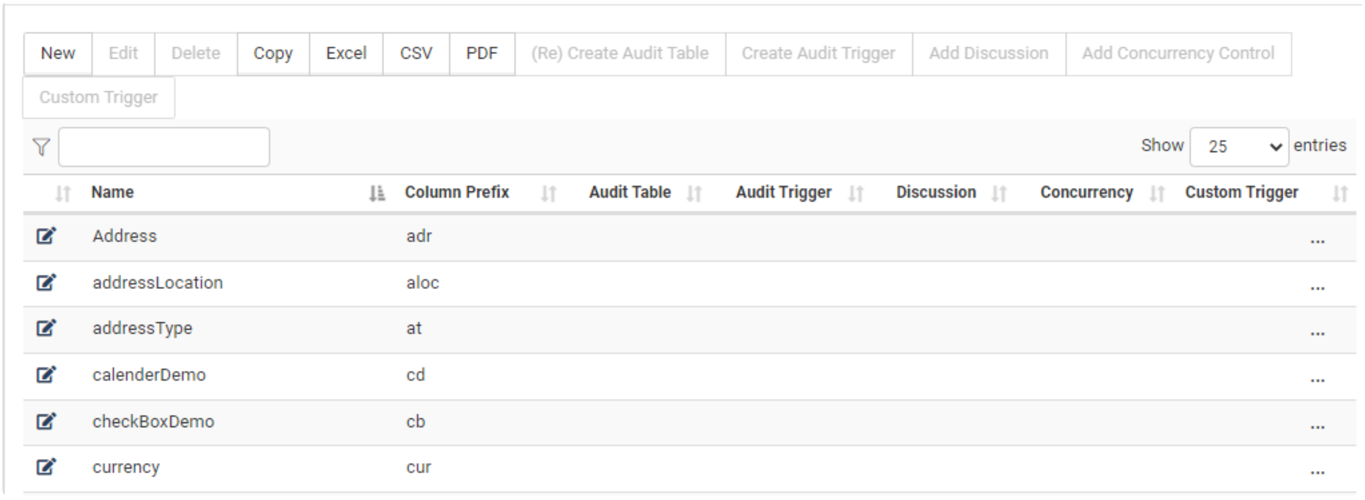
All the metadata information about users database is provided under this option. For example, name of database, creation date and available space etc. A snippet of an example database's info is given below:


View Database Information raghib

Name	raghib (Status: **ONLINE**)
Date of creation	**Jun 9, 2021 12:42 PM**
Current size	**120.0 MB**
Maximum size	250.0 GB
Available space	249.9 GB, 99%

Tables

Tables option provides user with searching, creating, editing, and exporting different data tables present in the database. It provides a detailed list of all the tables present. An example of tables present is listed below:



	Name	Column Prefix	Audit Table	Audit Trigger	Discussion	Concurrency	Custom Trigger
	Address	adr					...
	addressLocation	aloc					...
	addressType	at					...
	calenderDemo	cd					...
	checkBoxDemo	cb					...
	currency	cur					...

Sql Statements

Brixxbox allows users to create sql statements regarding their app's usability and save them. This option provides functionality to create, edit, and delete sql statements. It also lists all the present sql statements. In order to know more about managing them click here.

Sql Procedures

This option allows you to maintain SQL stored procedures for your Brixxbox Database. stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. It allows users with functionality to create, edit, and delete stored procedures. In order to learn more about it click here.

Sql Functions

Brixxbox allows users to write, edit, delete Sql functions. User only have to provide the body of a function, the function will be generated from it. In order to learn more about it click here. In order to learn more about it click here.

Sql Triggers

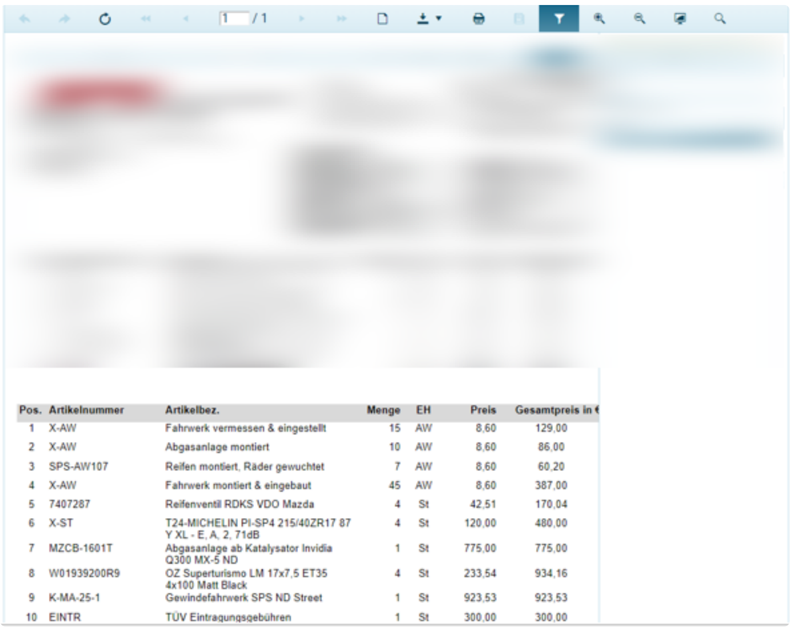
Triggers are such functions which run or get triggered when a certain event occurs. In this app, Brixxbox allows you to maintain DML triggers for your database tables. In order to learn more about it click here.

Data sources

Brixxbox allow users to work with new data sources using this option. It allows you add, update, and delete new data sources. Keep in mind that these data sources are different from the database option which is explained above.

Reports

In Brixxbox, user is able to add new reports. Brixxbox allows you to add html elements in your reports. This gives user to add anything to report page with respect to their needs. Brixxbox also allow user to install "telerik" report designer. It helps users to build, edit, and view reports. An simple example of a report is a "customerofferreport". A snapshot of example is given below for reference.


[redacted]

Pos.	Artikelnummer	Artikelbez.	Menge	EH	Preis	Gesamtpreis in
1	X-AW	Fahrwerk vermessen & eingestellt	15	AW	8,60	129,00
2	X-AW	Abgasanlage montiert	10	AW	8,60	86,00
3	SPS-AW107	Reifen montiert, Räder gewuchtet	7	AW	8,60	60,20
4	X-AW	Fahrwerk montiert & eingebaut	45	AW	8,60	387,00
5	7407287	Reifenventil RDKS VDO Mazda	4	St	42,51	170,04
6	X-ST	T24-MICHELIN PI-SP4 215/40ZR17 87 Y XL - E, A, 2, 71dB	4	St	120,00	480,00
7	MZCB-1601T	Abgasanlage ab Katalysator Invidia Q300 MX-5 ND	1	St	775,00	775,00
8	W01939200R9	OZ Superturismo LM 17x7,5 ET35 4x100 Matt Black	4	St	233,54	934,16
9	K-MA-25-1	Gewindefahrwerk SPS ND Street	1	St	923,53	923,53
10	EINTR	TÜV Eintragungsgebühren	1	St	300,00	300,00

In order to learn more about reports click here.

Custom messages

In this app, Brixxbox allows you to add custom messages. These messages can be used to portray warning, success or error state. For example, In case of successful order placement, we can show a thank you method to our user. In order to learn more about it click here.

Server Functions

Brixxbox is composed of maninly two parts: a javascript engine which works works in the browser, and web services which are present on server side. These are mainly used to get and post calls to web api's requests. On the front end, you can use "serverFunction" api in custom code to call different functions along with their parameters. Here is an example usage of serverFunction api.

```
let functionResponse = brixxApi.serverFunction("myFunction", {
    name: "Hallo"
});
console.log(functionResponse.functionResult);
```

In order to learn more about server functions click here.

Job Schedule

Brixxbox allows userss to automate their tasks which they want to perform on regular interval basis by using app job schedular. This minimizes user effort. For example, User want to update manager about current status of work on regular basis. One way is to do perform this task manually each day or user can simply use Brixxbox job schedular to automate this task once and for all. In order to learn more about App job click here.

Translations

This app provides functionality of logging messages that need top be translated from one language to another. For example you want to thank user after successfully placing an order. In order to learn more about App job click here.

Files

Brixxbox allows users to upload files from outside and it also stores files generated by different app in the workspace. There are 19 deiiferent types of documents are allowed in Brixxbox uptil now. In order to learn more about files or attachments click here

Document types

Brixxbox allows users to add different tpyes of documents in their workspace. In Brixxbox, each document is assigned a specific type for example invoice, list, bild etc. Document types lets you categorize attachments of anytype. The main properties of any attachemnt in Brixxbox are type id and name of document type. Document type is editable afterwards but id is not. In order to learn more about document type click here.

User Dashboards

This panel is used to configure which apps will be started for each user and role, when the user opens the their Brixxbox workspace. This app provides easy access to important applications without or little navigation. They enable your to become more productive. In order to learn more about document type click here.

Quickstart

Open the brixxbox app here

Client API Reference

brixxApi Functions

The brixxApi provides a set of functions for each brixxbox. These functions can be used inside brixxbox events, to interact with the brixxbox and modify the behaviour of the application.

Common Functions

- inviteUser(string email)

App Functions

- setToolbarButtonMode(string toolbarButtonId, string mode)
- addToolbarButton(toolbarButtonOptions)
- readOnlyMode(bool readOnly, options)
- excludeFromReadonly([])
- startScanner(string controlId, scanOptions)
- isUserInRole(string role)
- globalSearch(string searchTerm)
- getDate(Date date)
- getUserClaim(string claimName)
- getSessionToken()
- getBrowserToken()
- logAdd()
- enableNotifications()
- setToolbarButtonMode(string toolbarButtonId, string mode)
- addToolbarButton(toolbarButtonOptions)
- readOnlyMode(bool readOnly, options)
- excludeFromReadonly([])
- startScanner(string controlId, scanOptions)
- isUserInRole(string role)
- globalSearch(string searchTerm)
- getDate(Date date)
- getUserClaim(string claimName)
- getSessionToken()
- getBrowserToken()
- logAdd()
- enableNotifications()

Field Manipulation

- setFieldValue(string controlId, value) // Value can be of any javascript value type i.e. string, integer, Date etc.
- getFieldValue(string controlId)
- setLabelText(string controlId, string labelText)
- initAllControls()
- initControl()
- setControlUnmodifiedValue(string controlId, string valueToCompare) // Value can be of any javascript value type i.e. string, integer, Date etc.
- getControlUnmodifiedValue(string controlId[[[[markAllControlsAsUnModified|markAllControlsAsUnModified()]]]]
- getHtmlElement(string controlId)
- setVisibility(string controlId, bool visible, string subControl) // visible(default True) and subControlare optional parameters.
- setEnable(string controlId, bool visible, string subControl) // visible(default True) and subControl are optional parameters.
- triggerEvent(string eventName, string controlId)
- refresh(string controlId) // If used without parameter, refreshes all controls.
- refreshBadges()
- showTabPage(string controlId)setGridGrouping(string controlId, groupConfiguration) // groupConfiguration can either null, undefined or columnId.
- setGridAutoRefresh(string controlId, Number autoRefreshSeconds) // inputDate can be of Date type or a controlId of date/datetime control.
- getCalcDateTime(inputDate, accuracy) // inputDate can be of Date type or a controlId of date/datetime control and accuracy can be seconds(default), minutes, or hours.
- enableValidator(string controlId, string validatorName, bool enable) // enable can be true(default)/false.
- reInitValidation()
- setFieldUnit(string controlId, string unitName) // If no value is given for unitName then unit will be removed.
- setDecimalDigits(string controlId, Number value)
- addClassToGridRowCell(eventParameter, columnId, className)
- showRowDetailButton(Row, bool show)
- showRowDetailPanel(Row, bool show)
- clearGridSelection(string controlId)
- selectGridRows(string controlId, string columnId, value) // Value can be of any javascript value type i.e. string, integer, Date etc.
- unselectGridRows(string controlId, string columnId, value) // Value can be of any javascript value type i.e. string, integer, Date etc.
- toggleGridSelection(string controlId)
- isRowSelected(string controlId, line)
- switchTagControl(string controlId, bool editMode) // editMode is optional parameter.
- setFocus(string controlId, bool select)
- setTextColor(string controlId, string colorName) // colorName can be "default", "primary", "secondary", "success", "danger", or "warning".
- setFontWeight(string controlId, string fontWeight) // fontWeight can be normal or bold.
- setFontStyle(string controlId, string fontStyle) // fontStyle can be italic or normal.
- setBackgroundColor(string controlId, string colorName) // colorName can be "default", "primary", "secondary", "success", "danger", or "warning".
- disableGridHyperLinks(string controlId, bool disable)
- getSignatureImageBlob(string controlId, string type) // Type can be of png(default) or svg.
- cancelCalendarChanges(string controlId)
- addCalendarEventSource(string controlId, eventSource) //eventSource is a JSON Object.

- setFieldValue(string controlId, value) // Value can be of any javascript value type i.e. string, integer, Date etc.
- getFieldValue(string controlId)
- setLabelText(string controlId, string labelText)
- initAllControls()
- initControl()
-
  setControlUnmodifiedValue(string controlId, string valueToCompare) // Value can be of any javascript value type i.e. string, integer, Date etc.
- getControlUnmodifiedValue(string controlId[[[[markAllControlsAsUnModified|markAllControlsAsUnModified()]]]]
-

getHtmlElement(string controlId)
- setVisibility(string controlId, bool visible, string subControl) // visible(default True) and subControlare optional parameters.
- setEnable(string controlId, bool visible, string subControl) // visible(default True) and subControl are optional parameters.
- triggerEvent(string eventName, string controlId)
- refresh(string controlId) // If used without parameter, refreshes all controls.
- refreshBadges()
- showTabPage(string controlId)
- setGridGrouping(string controlId, groupConfiguration) // groupConfiguration can either null, undefined or columnId.
- setGridAutoRefresh(string controlId, Number autoRefreshSeconds) // inputDate can be of Date type or a controlId of date/datetime control.
- getCalcDateTime(inputDate, accuracy) // inputDate can be of Date type or a controlId of date/datetime control and accuracy can be seconds(default), minutes, or hours.
- enableValidator(string controlId, string validatorName, bool enable) // enable can be true(default)/false.
- reInitValidation()
- setFieldUnit(string controlId, string unitName) // If no value is given for unitName then unit will be removed.
- setDecimalDigits(string controlId, Number value)
- addClassToGridRowCell(eventParameter, columnId, className)
- showRowDetailButton(Row, bool show)
- showRowDetailPanel(Row, bool show)
- clearGridSelection(string controlId)
- selectGridRows(string controlId, string columnId, value) // Value can be of any javascript value type i.e. string, integer, Date etc.
- unselectGridRows(string controlId, string columnId, value) // Value can be of any javascript value type i.e. string, integer, Date etc.
- toggleGridSelection(string controlId)
- isRowSelected(string controlId, line)
- switchTagControl(string controlId, bool editMode) // editMode is optional parameter.
- setFocus(string controlId, bool select)
- setTextColor(string controlId, string colorName) // colorName can be "default", "primary", "secondary", "success", "danger", or "warning".
- setFontWeight(string controlId, string fontWeight) // fontWeight can be normal or bold.
- setFontStyle(string controlId, string fontStyle) // fontStyle can be italic or normal.
- setBackgroundColor(string controlId, string colorName) // colorName can be "default", "primary", "secondary", "success", "danger", or "warning".
- disableGridHyperLinks(string controlId, bool disable)
- getSignatureImageBlob(string controlId, string type) // Type can be of png(default) or svg.
- cancelCalendarChanges(string controlId)
- addCalendarEventSource(string controlId, eventSource) //eventSource is a JSON Object.

Printing and Documents

- print(string controlId) // Id of report control which is to be printed.
- printBlob(printDoc) // printDoc is a document which is to be printed.
- showBlob(string controlId, blobData)
- createReport(reportControlId, createOptions) // CreateOptions is a JSON object. It contains options like print, archive, saveFile etc. for report creation.

Data Storage

- loadRecord(string controlId)
- loadRecordById(Number recordId)
- loadConfigRecordById(string configName, Number recordId)
- deleteConfigRecordById(string configName, Number recordId)
- copyConfigRecordById(string configName, Number recordId, additionalValues) // additionalValues is a JSON object, it contains values need to be modified in target record.
- loadAndDisplayRecord(Number keyFieldId)

- loadAndDisplayRecordById(Number recordId)
- displayRecord(myRecord,string myKeyControlId)
- deleteRecord(options) // options is a JSON object which include properties id, noConfirmationMessage.
- validateInput()
- saveCurrentRecord(string configName, record) // record is a JSON like object.
- saveCurrentRecordWithoutEvents()
- saveConfigRecord(string configName, Number recordId)
- newRecord()
- newGridEntry(string gridControlId)
- executeStoredProcedure(string procedureName, procedureParameters, queryOptions) // procedureParameters and queryOptions are JSON objects.
- queryStoredProcedure(string procedureName, procedureParameters, queryOptions) // procedureParameters and queryOptions are JSON objects.
- refreshDataSource(string controlId) // Refreshes controlId. If used without parameters, it refreshes all controls with datasources.
- localValue(string keyName, string keyValue)
- sqlRead(string statementName, additionalParameters, queryOptions) // additionalParameters and queryOptions are JSON objects.
- sqlWrite(string statementName, additionalParameters, queryOptions) // additionalParameters and queryOptions are JSON objects.
- sqlReadValue(string statementName, additionalParameters, string columnName queryOptions)

Build in Tools

- composeEmail(emailOptions) // emailOptions is a JSON object, it contains options like to, cc, bcc, text, and subject etc which can be used to send an email.
- getGeoLocation()
- showWikiPage(string pageName, bool global) // If global is true, global wiki page is shown. Otherwise workspace wiki page is shown.
- showDiscussion(bool show) // Default show value is is true.
- showAttachments(bool show) // Default show value is is true.
- showMessage(messageOptions) // messageOptions is a JSON object.
- showMessageBox(messageBoxOptions) // messageBoxOptions is a JSON object.
- messageBox(messageBoxOptions) // messageBoxOptions is a JSON object.
- refreshAttachments()
- getAttachmentById(Number id)
- getAttachmentByFileName(string fileName)
- getAttachmentsForCurrentRecord()
- getAttachmentId(string fileName)
- getConfigRecordAttachmentIdByFileName(string appName, Number recordId, string fileName)
- downloadAttachments(downloadOptions) //downloadOptions is a JSON object. It iscludes resquestedIds list and file name for downloaded file.
- uploadAttachment(blob data,Number documentTypeId, string fileName) // documentTypeId and fileName are optional parameters.
- deleteAttachment(Number attachmentId)
- replaceText(string text, additionalReplacement) // additionalReplacement is a JSON object with key value pairs.
- callWebHook(string url, string message)
- getCustomMessage(string messageName, params, string targetLanguage) // params is a JSON object.
- getCustomSetting(string settingName)
- serverFunction(string funcName, funcParams, options) // funcParams and options are JSON objects and optional parameters.

Business Brixx

- DhlShiping
  - CreateShipment → Creates a DHL Shipping Label.
  - GetLabel → Shows the shipment Label again.
- DPDShipping
  - DPD Shipping

Starting brixxboxes

- startBrixxbox(startOptions) // startOptions is a JSON object with start parameters.
- addEventListener(string eventName, string controlId,function func) // ControlId is optional parameter.
- closeModal()
- isModal()
- createPublicAppUrl(options) //options is a JSON object.
- closePublicAppUrl(options) //options is a JSON object.
- logout()

Date and Time

- The moment.js Library

Variables

- userId
- recordId
- record
- isLoadingRecord
- isInitializing

Controls

- NumBox
- TextBox
- Button
- CheckBox
- ComboBox
- Grid
- TabControl
- TabPage
- Accordion
- Badge
- FullCalendar
- Camera
- Chart
- DateTimeBox
- DateBox
- DocViewer
- AppConfig
- FileImport
- FormGroup
- FormGroupRow
- GridConfig
- GroupBox
- HorizontalLine
- HtmlTable
- HtmlTemplate
- Image
- Label
- LinkLabel
- MultilineTextBox
- Report
- Row
- Scanner
- SignaturePad
- Tag
- TemplateGrid
- TemplateGridElement
- TimeBox
- Unit
- WedgeScanner
- Widget
- WidgetContainer
- WysiwygText

==

- Common Functions →
- App Functions →
- Field Manipulation → Functions which manipulate properties, values or behaviors of a control.
- Printing and Documents →
- Data Storage →
- Build in Tools →
- Business Brixx →
- Starting brixxboxes →
- Date and Time →
- Variables →

brixxApi Events

The brixxApi Events are used to react on certain events. This way you can customize the behaviour of the brixxbox to your own purpose. Some events will be raised befor the brixxbox executes their own logic for this event. In that case, you can return 'true' to avoid the brixxbox build in event. Otherwise the brixxbox will do its own logic after your event.

- onClick → When a control is clicked.
- onRowClick → When a grid row is clicked.
- onCellClick → When a grid cell is clicked.
- onChange → When a control content is changed.
- onRecordLoad → Before a Record is loaded.
- onRecordLoaded → After a Record is loaded.
- onRecordSave → Before a Record is saved.
- onRecordSaved → After a Record is saved.
- onRowSelectionChanged → After a the selected row of a grid has changed.
- onRowCreated → Modify a grid row. Apply a color for the row based on its values for example.
- onRecordNew → After initializing the Form. The new Record is not saved yet but can be modified with initial values.
- onRecordDelete → Before a record is deleted.
- onRecordDeleted → After a record was deleted.
- onChange → After a control value has changed.
- onKeyPress → When a key is pressed
- onKeyUp → When a key goes up
- onKeyDown → When a key goes down
- onModalClose → When a modal brixxbox app closes.
- onAppStart → When the app is started but befor it is initialized.
- onAppInitialized → When the app is initialized onAppStart has happend).
- onScan → When a code is scanned.
- onAttachmentsShow → Just before attachments panel opens.
- onFileImport → When a file is uploaded for import.
- onReturnFromModal → When a modal child app is closed, the parent will get this event
- onAttachmentsShow → When the sidebar for attachments opens
- onMailHistoryShow → When the sidebar for the mail history opens
- onSubDataRequest → When the subdatasource of a control is requested
- onChildAppClosed → When a childapp of your app closes.
- onTabShown → Fires when a tab page is changing to visible.
- onAttachmentDeleted → Triggers when the user deletes an attachment in the sidebar.
- onAttachmentsHide → This event occurs when the attachment sidebar gets closed.
- onDataTransform → The event can be used to transform a datasource line to a calender event.

- onEventClick → The event occurs the user clicks on one of the events.
- onEventChange → The event occurs the user moves or modifies an event.
- onTimeSelected → The event triggers when the users selects a time period in the calendar.

Functions

addCalendarEventSource

(Beta) Adds an event source object to the calendar

Parameters

1. controlId - The id of the calendar control
2. eventSource - the Event source object

Example Usages

```
app.addCalendarEventSource("myCal", {
   events: [
      {
         title: 'Sammple Event 1',
         start: '2020-12-08',
      },
      {
         title: 'Sammple Event 2',
         start: '2020-12-10',
      },
   ],
   id: 1,
   color: 'yellow',   // an option!
   textColor: 'black' // an option!
});
```

addClassToGridRowCell

This function is used to color a single cell in a grid, or highlight it.

Parameters

1. eventParameter - this is the special parameter you get in OnRowCreated
2. columnId - the name (id) of the column
3. className - the classnaem to set

Example Usages

Example inside OnRowCreated

```
    brixxApi.addClassToGridRowCell(eventArgs, "cordlnShippedQuantity", "success");
```

addEventListener

Adds an Event Listener to the brixxbox or one of its controls

Parameters

1. Events
2. Control Id (optional)
   - If the event is control specific (e.g. click on a buttton), this has to be the control id of that button
3. Function
   - The code that should be executed in case of this event
   - The first parameter is the brixxApi

Example Usages

```
brixxApi.addEventListener("onRecordSaved", function (brixxApi, eventArgs) {
    alert("Record saved in " + brixxApi.appName);
});
```

```
brixxApi.addEventListener("onClick", "myButton", function (/*event parameters are optional*/) {
    alert("Button clicked");
});
```

addToolbarButton

Add Toolbar Buttons with different functionalities.

As we have already discussed in documentation of SetToolbarButtonMode. These are several buttons that are provided by Brixxbox for each app like adding new record, saving new record etc. These buttons provides general functionality needed by most of the apps but in case your app needs extra functionality, Brixxbox also allows this. It is provided by brixxApi function **addToolbarButton**. It takes a JSON object as a parameter. This JSON contains all the properties that defines this custom button as well as the functionality provided by it. The properties include name, id, and function to execute on clicking this button etc. Adds a custom button to the app toolbar.

Example

```
    brixxApi.addToolbarButton({
            title: "Print Invoice",
            icon: "print",
            id: "printInvoiceToolbarButton",
            onclick: async function () {
                brixxApi.triggerEvent("click", "printInvoiceBtn");
            }
        });
```

Parameters

1. toolbarButtonOptions - It is a JSON Object. In JSON object we passes parameters in the form of key value arguments.It include properties like name, id, function, etc.

Example Usages

As we have already seen a minimal custom print button in above example. Lets now add extra properties like group, css class, and shortcut for our print button. Now we also want this button to be added in our app toolbar on initialization. For this purpose goto current app properties and add **onAppInitialized** event. Now we need to add following code as a custom code to this event.

```
    brixxApi.addToolbarButton({
            title: "Print Invoice",
            icon: "print",
            text: "Rechnung drucken",
            group: "invoice",
            id: "printInvoiceToolbarButton",
            cssClass: "btn-success",
            onclick: async function () {
                brixxApi.triggerEvent("click", "printInvoiceBtn");
            },
            shortcut: "Ctrl-Alt-P"
        });
```

Now when our app is initialized, we can see our custom button on the app toolbar and it should look like this.

We can also add different custom buttons to our app depending upon our own requirements.

brixxGPT

Calls the OpenAI integration in brixxbox

OpenAI API Key must be stored in the workspace settings!

OpenAI API Key must

Parameter

gptOptions - Json object with parameters.

- system - to set the stage. A promt to tell the model, how to behave
- user - User input, examples and a query. This is the user promt

Example Usages

```
let response = app.brixxGPT({
    system: "Reply in a HTML snipped Text, that will be inserted in an existing div",
    user: "How to create a for loop in JS?"
});

// "output" is a brixxbox htmlTemplate control
$(app.getHtmlElement("output")).html(response);
```

businessBrixx

Business Brixxes provides special features for business usecases. Each Brixx has an individual set of parameters plus businessfunctions.

Parameters

```
functionName - the name of the businessBrixx module
methodName - some businessBrixx functions provide several operations
silentMode - if you set this to true, the brixxbox will not show a message if something goes wrong. You will still get the error
```

Example Usages

```
  let result = await brixxApi.businessBrixx({
     //Common Parameters for all functions
     functionName:"DhlShiping",
     methodName: "GetLabel",
     silentMode: true,  //false is default. If set to true, the brixxbox will not show an error message box. it is up to you to

     //Individual paremeters for each business case. DHL Label as an example here.
     shipmentNumber: app.getFieldValue("cordShipmentNumber") //We saved this number before and stored it to our Order
  });
  window.open(result.labelUrl, "_blank");
```

Business Brixx

- DhlShiping
  - CreateShipment → Creates a DHL Shipping Label.
  - GetLabel → Shows the shipment Label again.
- DPD Shipping
  - CreateShipment → Create a DPD Shipping Label.
- UPS Shipping
  - CreateShipment → Create a UPS Shipping Label.
  - CancelShipment → Removes existing Shipment
  - GetLabel → Shows the shipment Label again.
- InventoryManagement → Does Inventory Movements within an ERP Setup
  - Move
- CurrencyConverter
  - Convert → Converts a currency to annother currency.
- Datev Export
  - DatevExport → This function exports a Datev CSV file for a given timeframe.
    - sqlDataRead Example
    - sqlMainData Example
    - sqlGetCounter Example
- GS1 Code Splitter
  - Split → Splits a barcode (or 2d code) into its GS1 AIs an Values.
- Diamant
  - DiamantBusinessBrixx

DHL Shiping

- CreateShipment → Creates a DHL Shipping Label.
- GetLabel → Shows the shipment Label again.

CreateShipment

Creates a DHL Shipping Label.

If shipService is omitted the default value ""V01PAK" will be set.

Example Usages

```
let result = await brixxApi.businessBrixx({
   functionName:"DhlShiping",
   methodName: "CreateShipment",
   weightInKG: "4",
   lengthInCM: "20",
   heightInCM: "20",
   widthInCM: "10",
   shipService: "V01PAK",
   recipientEmailAddress: "info@acme.com",
      senderAddress:{
      name1: app.getFieldValue("cordCompanyId.cusName"),
      streetName: app.getFieldValue("cordCompanyId.cusStreet"),
      streetNumber: app.getFieldValue("cordCompanyId.cusStreetNumber"),
      zip: app.getFieldValue("cordCompanyId.cusZip"),
      city: app.getFieldValue("cordCompanyId.cusCity"),
      countryCode: "DE",
      },
      receiverAddress:{
      name1: app.getFieldValue("cordCustomerId.cusName"),
      streetName: app.getFieldValue("cordCustomerId.cusStreet"),
      streetNumber: app.getFieldValue("cordCustomerId.cusStreetNumber"),
      zip: app.getFieldValue("cordCustomerId.cusZip"),
      city:app.getFieldValue("cordCustomerId.cusCity"),
      countryCode: "DE",
      },
   });
console.log("Shipment Number: " + result.shipmentNumber)
window.open(result.labelUrl, "_blank");
```

GetLabel

Shows the shipment Label again.

Example Usages

```
let result = await brixxApi.businessBrixx({
   functionName:"DhlShiping",
   methodName: "GetLabel",
   shipmentNumber: app.getFieldValue("cordShipmentNumber")
});
window.open(result.labelUrl, "_blank");
```

DPD Shiping

DPDCreateShipment -> Creates a DPD Shipping Label

CreateShipment

Creates a DPD Shipping Label

For accessing the DPD-Shipping brixxbox interface the DPD services need the following setup values (configuration -> settings).

- **DPDUserCredentialsCloudUserID**
- **DPDUserCredentialsToken**
- **DPDPartnerCredentialsName**
- **DPDPartnerCredentialsToken**

If no settings are available, a sample label will be generated.

Incomming Information

To create labels three groups of information are available:

- **1. OrderSettings** with the following fields (If the group is not present default values will be assumed)
  - **language:** Values „de_DE“ or „en_EN“ are allowed. If nothing is set „de_DE“ will be assumed.
  - **ShipDate:** if not available will be set to „today“
  - **LabelSize:** „PDF_A4“ or „PDF_A6“. If nothing is set „PDF_A4“ will be assumed.
  - **LabelStartPosition:** “UpperLeft“, „UpperRight“, „LowerLeft“, „LowerRigh“
- **2. ShipAddress:**
  - Company
  - Gender
  - Salutation
  - FirstName
  - LastName
  - Name
  - Street
  - HouseNo
  - ZipCode
  - City
  - Country
  - State
  - Phone
  - Mail
- **3. ParcelData:**
  - **YourInternalID:** should be set to your own reference (e.g. order no). if no value is provided a timestamp will be set.
  - Content
  - **Weight:** in kg
  - Reference1
  - Reference2
  - **ShipService:** set to a valid DPD-Service. If nothing is set “Classic” will be assumed.

Detailed informationen for field usage can be found at
https://esolutions.dpd.com/dokumente/DPD_Cloud_Service_Webservice_Dokumentation_DE.pdf

Returning Information

- In case of **success (status 200)** the following fields are returned
  - **ParcelNo:** The DPD generated ID. This will be used for further references to this shipment.
  - **YourInternalID:** The same ID that was provided for the request
  - **TimeStamp:** an informational time stamp when the request was processed
  - **LabelData:** this is a blob response. It contains the actual label in PDF format and can directly be used for further processing. E.g. “brixxApi.printBlob(LabelPDF);”
- In case of **error (status 400)** the following fields are returned:
  - **TimeStamp:** an informational time stamp when the request was processed
  - **ErrorDataList:** an array with detailed information what went wrong. Every error will show the following fields
    - **ErrorID:** an internal error no
    - **ErrorCode:** an internal error code
    - **ErrorMsgShort:** Error information short version
    - **ErrorMsgLong:** Error information long version

Example Usages

```
/*----------------------------------------------------------------------------------------------
/This example is minimalistic and will create a sample label.
/You need to check which fields are relevant for your process and assign the values accordingly.
/----------------------------------------------------------------------------------------------*/

var result = await brixxApi.businessBrixx({
      functionName:"DPDShipping",
      methodName: "CreateShipment",
          ShipAddress : {
                Company: 'Mustermann AG',
                Name: 'Max Mustermann',
                Street: 'Wailandtstr.',
                HouseNo: '1',
                ZipCode: '63741',
                City: 'Aschaffenburg',
                Country: 'DEU'
          },
          ParcelData : {
                Reference1: 'Customer email',
                Content: 'Order number',
                Weight: '13.5'
      }
      });
console.log("Parcel Number: " + result.ParcelNo)
brixxApi.printBlob(result.LabelData);


/*----------------------------------------------------------------------------------------------
/This example uses all fields
/You need to check which fields are relevant for your process and assign the values accordingly.
/----------------------------------------------------------------------------------------------*/
var result = await brixxApi.businessBrixx({
      functionName:"DPDShipping",
      methodName: "CreateShipment",
                        OrderSettings : {
                                language: 'de_DE',
                                LabelSize : 'PDF_A4',
                                LabelStartPosition : 'UpperLeft',
                                ShipDate : app.getFieldValue("SomeFieldValue")
                          },
                        ShipAddress : {
                                Company: app.getFieldValue("SomeFieldValue"),
                                Gender: app.getFieldValue("SomeFieldValue"),
                                Salutation: app.getFieldValue("SomeFieldValue"),
                                FirstName: app.getFieldValue("SomeFieldValue"),
                                LastName: app.getFieldValue("SomeFieldValue"),
                                Name: app.getFieldValue("SomeFieldValue"),

                                Street: app.getFieldValue("SomeFieldValue"),
                                HouseNo: app.getFieldValue("SomeFieldValue"),
                                ZipCode: app.getFieldValue("SomeFieldValue"),
                                City: app.getFieldValue("SomeFieldValue"),
                                Country: app.getFieldValue("SomeFieldValue"),
                                State: '',
                                Phone: app.getFieldValue("SomeFieldValue"),
                                Mail: app.getFieldValue("SomeFieldValue")
                          },
                        ParcelData : {
                                Reference1: app.getFieldValue("SomeFieldValue"),
                                Reference2: app.getFieldValue("SomeFieldValue"),
                                Content: app.getFieldValue("SomeFieldValue"),
                                Weight: app.getFieldValue("SomeFieldValue"),
                                YourInternalID: app.getFieldValue("SomeFieldValue"),
                                ShipService: app.getFieldValue("SomeFieldValue")
                          }
      });
console.log("Parcel Number: " + result.ParcelNo)
brixxApi.printBlob(result.LabelData);
```

UPS Shiping

Contents

Business Brixx UPS Shipping

For accessing the UPS-Shipping brixxbox interface the UPS services need the following setup values (configuration -> settings).

- **UPSCustomerID** Your customer number with UPS
- **UPSUsername** A valid User for accessing the services
- **UPSPassword** The corresponding password
- **UPSAccessLicenseNumber** A license/access key provided by UPS on this page https://www.ups.com/upsdeveloperkit

If no settings are available, a sample label will be generated.

Business Brixx UPS Shipping -> CreateShipment

Creates a UPS shipping label

Incomming Information

To create labels four groups of information are available:

- **1. Settings** with the following fields (If the group/field is not present default values will be assumed)
  - **ServiceCode:** default value "11" UPS Standard. Valid values according to UPS service codes.
  - **MetricMeasurements:** "1" sets to true. This is the default. Measurements will be metric e.g. kg, cm or nonmetric for lbs, inch
  - **Language:** Language code according to UPS. e.g. deu, eng, spa. this field will be set to user language by default. If the user language is not available for UPS, "eng" will be used.
- **2. Shipper:**
  - Name
  - Street
  - HouseNo
  - ZipCode
  - City
  - CountryCode
  - StateProvinceCode
  - AttentionName
  - Phone
  - EMail
  - FaxNumber
  - TaxIdentificationNumber
- **3. ShipAddress:**
  - Name
  - Street
  - HouseNo
  - ZipCode
  - City
  - CountryCode
  - StateProvinceCode
  - AttentionName
  - Phone
  - EMail
  - FaxNumber
  - TaxIdentificationNumber
- **4. ParcelData:**
  - Description
  - **PackageType:** Package type according to UPS. If no value is given, the default will be set to: "02" Customer Supplied Package
  - Weight
  - Height
  - Length
  - Width

Returning Information

- In case of **success (status 200)** the following fields are returned
  - **TrackingNumber:** The UPS generated ID. This will be used for further references to this shipment.
  - **LabelData:** this is a blob response. It contains the actual label in GIF format and can directly be used for further processing. E.g. “brixxApi.printBlob(LabelData);”
- In case of **error (status 400)** the following fields are returned:
  - **error:** a text showing the first error. E.g. “console.log(result.error);”

Example Usages

```
var result = await brixxApi.businessBrixx({
      functionName:"UPSShipping",
      methodName: "CreateShipment",
                        Settings : {
                                Language: 'deu',
                                MetricMeasurements: 1,
                                ServiceCode: "11"
                          },
                        Shipper : {
                                Name: 'Brixxbox GmbH',
                                Street: 'Husarenstraße',
                                HouseNo: '34a',
                                ZipCode: '41836',
                                City: 'Hückelhoven',
                                CountryCode: 'DE'
                          },
                        ShipAddress : {
                                Name: 'Brixxbox GmbH',
                                Street: 'Husarenstraße',
                                HouseNo: '34b',
                                ZipCode: '41836',
                                City: 'Hückelhoven',
                                CountryCode: 'DE',
                                Phone: '+49 1234 5678-9',
                                Mail: 'm.mustermann@mustermann.com'
                          },
                        ParcelData : {
                                Description: 'Product name',
                                Height : 10,
                                Length : 20,
                                Width  : 15,
                                Weight: '4.5',
                                ServiceCode: '9'
                          }
      });

console.log("Tracking Number: " + result.TrackingNumber);
brixxApi.printBlob(result.LabelData);
```

CancelShipment

To cancel an existing shipment only the **TrackingNumber** is needed.

```
var result = await brixxApi.businessBrixx({
      functionName:"UPSShipping",
      methodName: "CancelShipment",
         TrackingNumber: app.getFieldValue("ExistingTrackingNumber")
      });
```

GetLabel

In case the label is needed again it can be retrieved with an existing **TrackingNumber**. The result will show LabelData (see **CreateShipment results** #Returning_Information for details)

```
var result = await brixxApi.businessBrixx({
      functionName:"UPSShipping",
      methodName: "GetLabel",
         TrackingNumber: app.getFieldValue("ExistingTrackingNumber")
      });
```

Inventory Movement

Stores an inventory movement in a table with a certain structure. The Function handles database transactions and generates movement ids.

Mandatory fields: imItemId, imQuantity

Example Usages

```
    await brixxApi.businessBrixx({
         functionName:"InventoryMovement",
         movementTable:"inventoryMovement",
         movements:[
     {
                 imQuantity: app.getFieldValue("quantityToShip") * -1,
                 imAddressId: app.getFieldValue("companyId"),
                 imItemId: app.getFieldValue("itemId"),
         imImei: app.getFieldValue("imei"),
         },
         {
                 imQuantity: app.getFieldValue("quantityToShip"),
                 imAddressId: app.getFieldValue("customerId"),
                 imCustomerUserId: app.getFieldValue("customerUserId"),
                 imItemId: app.getFieldValue("itemId"),
         imCustomerOrderLineId: app.getFieldValue("orderLineId"),
         imImei: app.getFieldValue("imei"),
         },
         ]
     });
```

Currency Conveter

Converts a currency to another currency

Mandatory fields: fromCurrencySymbol, toCurrencySymbol

Example Usages

```
let rate = await brixxApi.businessBrixx({
    functionName:"CurrencyConverter",
    methodName: "Convert",
    fromCurrencySymbol: "EUR",
    toCurrencySymbol: "USD"
});
console.log(rate)
```

Datev Export

This function exports a Datev CSV file for a given timeframe. The fuction needs 3 customized SQL scripts to process the data.

As this function calls customized SQL scripts, it will add SQL parameters from the function call to the SQL calls.

- @company
- @dateFrom
- @dateUntil

Those parameters can be used within the SQL statements.

Needed output columns from SQL scripts

- sqlDataRead script
    - Auftragsnummer
    - Artikelnummer
    - ArtikelRecordId
    - Paragraph13b
    - NettoWert
    - BruttoWert
    - Debitorenkonto
    - Kreditorenkonto
    - Sachkonto
    - SteuerID
    - Buchungsschluessel
    - Buchungstext
    - Rechnungsdatum
    - Rechnungsnummer
    - Leistungsdatum
    - AttachementId
    - BuchungszaehlerTabelle (nvarchar that defines the tabel, where the booking counter is stored)
    - BuchungszaehlerSpalte (nvarchar that devines the column for the booking counter)
    - IstAutomatikKonto
    - IstInnerbetrieblich
- sqlMainData script
    - Kundennummer
    - Konto
    - Firmenname
    - Nachname
    - Vorname
    - Anrede
    - Adresstyp
    - Strasse
    - Hausnummer
    - PLZ
    - Stadt
    - Zusatzinformation
- sqlGetCounter script
    - This script needs to return the actual booking counter as number.

SQL script examples

- sqlDataRead Example
- sqlMainData Example
- sqlGetCounter Example

Example Usages

```
brixxApi.businessBrixx({
   functionName:"DatevExport",
   sqlDataRead:"getDatevData",
   sqlMainData:"getDatevMainData",
   sqlGetCounter:"getDatevBookingCounter",
   exportFromDate: moment().subtract(2,'months').startOf('month').format('YYYY-MM-DD'),
   exportUntilDate: moment().subtract(2,'months').endOf('month').format('YYYY-MM-DD'),
   consultantNumber: 1234567,
   clientNumber: 12345,
   company: 1,
   isDownload: true
});
```

Datev Export SqlDataRead Example

Example for a getDatevData script

This is the main script, which is responible for collection all the data, which needs to be exported.

Available Parameters

1. @company - If you work with multiple companies, you have to export each company by it's own.
2. @dateFrom - Start date for the export
3. @dateUntil - End date for the export

Example Usages

```sql
SELECT
   head.id AS Auftragsnummer,
   ol.cordlnItemId AS Artikelnummer,
   ol.id AS ArtikelRecordId,
   ol.cordlnParagraph13b AS Paragraph13b,
   ol.cordlnNetOrderValue AS NettoWert,
   ol.cordlnGrossOrderValue AS BruttoWert,
   adr.adrDebitor AS Debitorenkonto,
   0 AS Kreditorenkonto,
   cordlnSalesAccount AS Sachkonto,
   adr.adrTaxId AS SteuerID,
   cordlnTaxKey as Buchungsschluessel,
   '' AS Buchungstext,
   head.cordInvoiceDate AS Rechnungsdatum,
   head.cordInvoiceNumber AS Rechnungsnummer,
   head.cordShippingDate AS Leistungsdatum,
   atta.id AS AttachementId,
   'customerOrder' AS BuchungszaehlerTabelle,
   'cordDatevExportId' AS BuchungszaehlerSpalte,
   CASE WHEN fiac.finaccIsAccountSplitting = 1 THEN fiacsplit.facsIsAutomaticAccount ELSE fiac.finaccIsAutomaticAccount END AS I
   head.cordInterCompanyOrder as IstInnerbetrieblich

FROM dbo.bbv_customerOrderLine AS ol
LEFT JOIN [dbo].[customerOrder] AS head ON head.id = ol.cordlnOrderId
LEFT JOIN address AS adr ON adr.id = head.cordAddressId
LEFT JOIN brixx_Attachments AS atta ON atta.id in (select AttachmentId from brixx_AttachmentLinks where RecordId = head.id AND T
LEFT JOIN financialAccount AS fiac ON fiac.finaccAccountNumber = cordlnSalesAccount
LEFT JOIN financialAccountSplitting AS fiacsplit ON fiacsplit.facsAccountId = fiac.id AND fiacsplit.facsSplitAccount = fiac.fina
WHERE ol.cordlnOrderId IN
(
   SELECT id FROM [dbo].[customerOrder]
   WHERE cordShippingDate >= @dateFrom AND cordShippingDate <= @dateUntil
   AND cordCompanyId = @company
)
AND ol.cordlnNetOrderValue > 0
AND head.cordStatusId >= 4

UNION

SELECT
   head.id AS Auftragsnummer,
   ol.sordlnItemId AS Artikelnummer,
   ol.id AS ArtikelRecordId,
   ol.sordlnParagraph13b AS Paragraph13b,
   ol.sordlnNetOrderValue AS NettoWert,
   ol.sordlnGrossOrderValue AS BruttoWert,
   0 AS Debitorenkonto,
   adr.adrKreditor AS Kreditorenkonto,
   sordlnitmProcurementAccount AS Sachkonto,
   adr.adrTaxId AS SteuerID,
   sordlnTaxKey AS Buchungsschluessel,
   '' AS Buchungstext,
   head.sordGoodsReceiptDate AS Rechnungsdatum,
   head.sordInvoiceNumber AS Rechnungsnummer,
   head.sordGoodsReceiptDate AS Leistungsdatum,
   atta.id AS AttachementId,
   'supplierOrder' AS BuchungszaehlerTabelle,
   'sordDatevExportId' AS BuchungszaehlerSpalte,
   CASE WHEN fiac.finaccIsAccountSplitting = 1 THEN fiacsplit.facsIsAutomaticAccount ELSE fiac.finaccIsAutomaticAccount END AS I
   head.sordInterCompanyOrder as IstInnerbetrieblich

FROM dbo.bbv_supplierOrderLine AS ol
LEFT JOIN [dbo].[supplierOrder] AS head ON head.id = ol.sordlnOrderId
LEFT JOIN address AS adr ON adr.id = head.sordAddressId
LEFT JOIN item AS itm on itm.id = ol.sordlnItemId
LEFT JOIN brixx_Attachments AS atta ON atta.id in (select AttachmentId from brixx_AttachmentLinks where RecordId = head.id AND T
LEFT JOIN financialAccount AS fiac ON fiac.finaccAccountNumber = ol.sordlnitmProcurementAccount
LEFT JOIN financialAccountSplitting AS fiacsplit ON fiacsplit.facsAccountId = fiac.id AND fiacsplit.facsSplitAccount = fiac.fina
```

```
WHERE ol.sordlnOrderId IN
(
   SELECT id FROM [dbo].[supplierOrder]
   WHERE head.sordGoodsReceiptDate >= @dateFrom AND head.sordGoodsReceiptDate <= @dateUntil
   AND sordCompanyId = @company
)
AND ol.sordlnNetOrderValue > 0
AND head.sordStatusId = 5
```

Datev Export sqlMainData Example

This is the script, which is responsible for collection all the address data, which needs to be exported.

Available Parameters

1. @company - If you work with multiple companies, you have to export each company by it's own.
2. @dateFrom - Start date for the export
3. @dateUntil - End date for the export

Example Usages

```
SELECT adr.id AS Kundennummer,
      CASE WHEN adrIsCustomer = 1 THEN adr.adrDebitor ELSE adr.adrKreditor END AS Konto,
      adr.adrName AS Firmenname,
      adr.adrLastName as Nachname,
      adr.adrFirstName as Vorname,
      '' AS Anrede,
      'STR' AS Adresstyp,
      adrLoc.alocStreet AS Strasse,
      adrLoc.alocStreetNumber AS Hausnummer,
      adrLoc.alocZip AS PLZ,
      adrLoc.alocCity AS Stadt,
      adrLoc.alocAdditionalInfo AS Zusatzinformation
 FROM [dbo].[address] AS adr
 LEFT JOIN addressLocation AS adrLoc ON adrLoc.id = adr.adrDefaultInvoiceAddress
 LEFT JOIN country AS country ON country.id = adrLoc.alocCountryId
WHERE adr.adrIsCustomer = 1 OR adr.adrIsSupplier = 1
```

Datev Export sqlGetCounter Example

This is the main script, which is responsible for collection all the data, which needs to be exported.

Available Parameters

1. @company - If you work with multiple companies, you have to export each company by it's own.
2. @dateFrom - Start date for the export
3. @dateUntil - End date for the export

Example Usages

```sql
SELECT MAX(DatevExportId) FROM
(
   SELECT ISNULL(MAX(cordDatevExportId), 0) AS DatevExportId FROM customerOrder WHERE cordCompanyId = @company
   UNION ALL
   SELECT ISNULL(MAX(sordDatevExportId), 0) AS DatevExportId FROM supplierOrder  WHERE sordCompanyId = @company
) AS sub
```

GS1 Code Splitter

Splits a GS1 code into its AIs and values

This is usually used in the onScan event where you use the plain scan result as the input. The fnc1Char parameter is optional and only used, if the scanner sends an different (visible) character instead of the default fnc1 (CHAR29).

Mandatory fields: gs1Code

Example Usages

```
let result = await brixxApi.businessBrixx({
    functionName:"Gs1Splitter",
    methodName: "Split",
    gs1Code: "01123456789012341721052110998 8"
});
console.log(result)
```

Example Usages with Split Char (FNC1)

Especially with keyboard wedge scanners, it is often usefull to use a visible character as the FNC1 char.

```
let result = await brixxApi.businessBrixx({
    functionName:"Gs1Splitter",
    methodName: "Split",
    gs1Code: "011234567890123417210521109988X211234",
    fnc1Char: 'X'
});
console.log(result.applicationIdentifiers['10'])
```

The result variable in this case:

```
{
   "applicationIdentifiers":{
      "10":"9988",
      "17":"210521",
      "21":"1234",
      "01":"12345678901234",
      "01.packind":"1",
      "01.iln":"2345678",
      "17.date":"20210521"
   }
}
```

Diamant Connector

This brixxbox module allows to retrieve data from the Diamant Accounting Software and post data to it

Get Method

The Method Retrieves the data specified with **dataType** (like Address, CostCenter and so on). The **searchParams** and **fieldMatch** JSON properties are valid for all methods, but the fields mentioned in it are depending on the datatype used. For example: **postcode** is a valid **searchParam** for addresses, but not for cost centers.

The Get method will return the data it fetched from Diamant, so you get an array of JSON objects. The fieldMatch prameter will tell the brixxbox to change field names, so that it is more useful in an brixxbox environment.

Example Usages Get

This will retrieve all addresses from diamant. Field names are the original diamand field names

```
let result = await brixxApi.businessBrixx({
    functionName:"Diamant",
    methodName: "Get",
    company: "9018",
    dataType: "Address",
});
console.log(result)
```

This will retrieve all addresses with Postcode 61118 from diamant and return the result. The fields **postcode** is changed to **adrZip** by the brixxbox and **name1** is changed to **adrName**. All other fields are ignored and are **not** in the result set.

```
let result = await brixxApi.businessBrixx({
    functionName:"Diamant",
    methodName: "Get",
    dataType: "Address",
    searchParams:{
        postcode: "61118"
    },
    fieldMatch:{
        postcode: "adrZip",
        name1: "adrName"
    },
});
console.log(result)
```

ImportTable Method

This method allows you to retrieve the data and insert it into the database. Query parameters work the same and are optional but you must user fieldMatch to assign diamant names to brixxbox field names

Example Usages Import Insert

This will retrieve all addresses from diamant and insert it into the table **address** the table will be wiped right before the import.

```
let result = await brixxApi.businessBrixx({
    functionName:"Diamant",
    methodName: "ImportTable",
    dataType: "Address",
    tableName: "address",
    wipeTable: true,
    fieldMatch:{
        postcode: "adrZip",
        name1: "adrName"
    },
});
console.log(result)
```

Example Usages Import Upsert

This will retrieve all addresses from diamant and update the records with the same id as in fieldMatch.

```
let result = await brixxApi.businessBrixx({
    functionName:"Diamant",
    methodName: "ImportTable",
    dataType: "Address",
    tableName: "address",
    upsert: true,
    wipeTable: true,
    fieldMatch:{
        key: "adrKey,id",
        postcode: "adrZip",
        name1: "adrName"
    },
});
console.log(result)
```

Supported dataTypes

More types can be added at request.

Request Data

- Address (Adressen)
- CostCenter (Kostenstellen)
- GenLedgAccount (Sachkonten)
- CostObject (Kostenträger)
- PrimCostElement (Primärkostenarten)
- Project (Projekte)
- Company (Mandanten)
- Customer(Konten)
- Vendor (Vendor)
- Posting (Buchungen)

Transctions

- OpenStack
- Transaction
- CloseStack

Diamant OpenStack

Opens a booking stack and returns the number to the client

Example Usages Get

```
 let result = await app.businessBrixx({
    functionName:"Diamant",
    methodName: "OpenStack",
});
app.setFieldValue("myBookungStackNumber", result.data); //result.data is the stackNumber
```

Diamant Transaction

Creates a transaction in diamant

Example Usage

```
let result = await app.businessBrixx({
    functionName:"Diamant",
    methodName: "Transaction",
    stackNumber: app.getFieldValue("myBookungStackNumber"), //if stackNumber is not set or 0, the brixxbox will automatically cre
    bookingDate: "2021-12-23",
    type: "AR",
    number: "12345678",
    currency: "EUR",
    debitAccount: "20003000",
    debitValue: 3334,
    text: "Volkers Buchung",
    dueDate: "2022-04-30",
    creditAccount: "409000",
    creditValue: 3334,
    taxCode: 200,
    creditCostCenter: "21000",
    creditBusinessUnit: "100"
});
console.log(result);
```

Diamant CloseStack

Closes a booking stack that qas previously open by OpenStack

Example Usages Get

```
 let result = await app.businessBrixx({
    functionName:"Diamant",
    methodName: "CloseStack",
    stackNumber: app.getFieldValue("myBookungStackNumber")
});
```

cancelCalendarChanges

(Beta) Cancels the drag or edit operation that triggered the OnEventChange event and moves all events back to their origin position. Must be used inside the OnEventChange event.

Parameters

1. controlId - The id of the calendar control

Example Usages

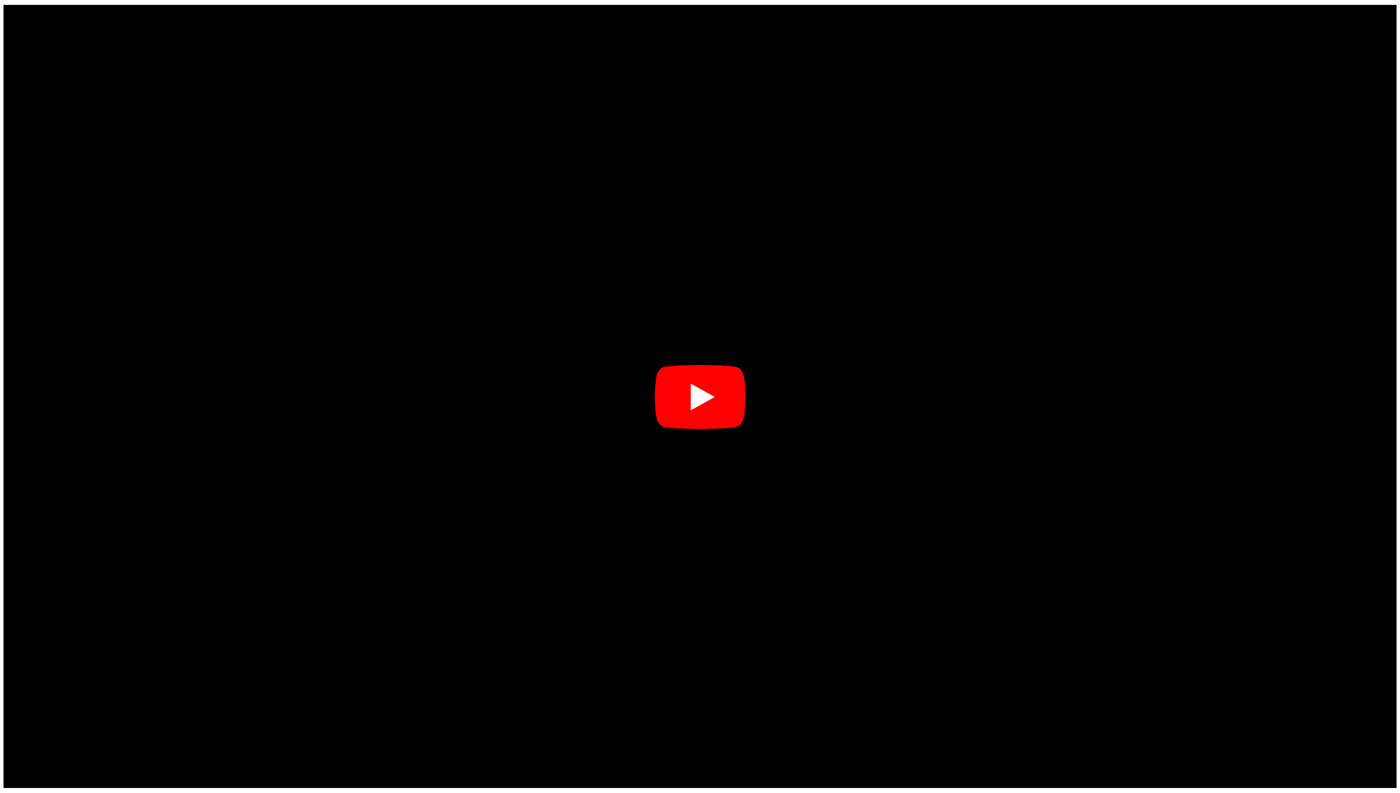
```
    brixxApi.cancelCalendarChanges("myCalendar");
```

callWebHook

Calls a webhook to trigger a system to do anything like posting a message to a teams channel.

Parameters

1. url - The url of the teams or slack channel, see the video to learn how to get a WebHook url.



2. the message you want to send. That depends on the service, you want to post to. here are some examples for Microsoft Teams(https://docs.microsoft.com/de-de/microsoftteams/platform/webhooks-and-connectors/how-to/connectors-using), but it can be as easy as seen in exampel 1

Example Usages

Plain text message

```
brixxApi.callWebHook(myUrl, {
   text: "Hello World"
});
```

full styled message with actions

```
brixxApi.callWebHook(myUrl, {
   "@type": "MessageCard",
   "@context": "http://schema.org/extensions",
   "themeColor": "006E7E",
   "summary": "A new message from brixxbox",
   "sections": [{
      "activityTitle": "![TestImage](https://47a92947.ngrok.io/Content/Images/default.png)A new message from brixxbox",
      "activitySubtitle": "Volkers Sandbox",
      "activityImage": "https://teamsnodesample.azurewebsites.net/static/img/image5.png",
      "facts": [{
         "name": "Assigned to",
         "value": app.userId
     }],
     "markdown": true
   }],
   "potentialAction": [{
      "@type": "ActionCard",
      "name": "Add a comment",
      "inputs": [{
         "@type": "TextInput",
         "id": "comment",
         "isMultiline": false,
         "title": "Add a comment here for this task"
     }],
     "actions": [{
         "@type": "HttpPOST",
         "name": "Add comment",
         "target": "http://..."
     }]
   }, {
     "@type": "ActionCard",
     "name": "Set due date",
     "inputs": [{
         "@type": "DateInput",
         "id": "dueDate",
         "title": "Enter a due date for this task"
     }],
     "actions": [{
         "@type": "HttpPOST",
         "name": "Save",
         "target": "http://..."
     }]
   }, {
     "@type": "ActionCard",
     "name": "Change status",
     "inputs": [{
         "@type": "MultichoiceInput",
         "id": "list",
         "title": "Select a status",
         "isMultiSelect": "false",
         "choices": [{
             "display": "In Progress",
             "value": "1"
         }, {
             "display": "Active",
             "value": "2"
         }, {
             "display": "Closed",
             "value": "3"
         }]
     }],
     "actions": [{
         "@type": "HttpPOST",
         "name": "Save",
         "target": "http://..."
     }]
   }]
});
```

changeHelpText

Changes an existing help text for a control at runtime.

Parameters

1. controlId => Id of the control
2. helpText => the new text, you want to set or empty to remove an existing text

Example Usages

```
    brixxApi.changeHelpText("myControl", "Hello World");
```

clearGridSelection

Unselects all rows of a data grid.

Parameters

1. controlId - id of the grid control

Example Usages

```
    brixxApi.clearGridSelection(myGrid);
```

cloudPrint

Sends a PDF to a printer, that is connected by the BrixxboxCloudGateway. You can sent a document to a cloudprinter by using createReport as well

Parameters

A json object

Example Usages

Print an existing attachment

```
    brixxApi.cloudPrint({
       printerName: "HP LaserJet", //Printer NAme
       attachmentId: "1", //Attachment id
    });
```

Print an existing attachment with multiple copies

```
    brixxApi.cloudPrint({
       printerName: "HP LaserJet",
       attachmentId: "1",
       copies: 5 // 1 copy is the default. Needs "cloudPrint Gateway" installer Version 1.0.1 or newer
    });
```

Print a new created report without archiving

```
    let myReport = await brixxApi.createReport("myReport");
    brixxApi.cloudPrint({
       printerName: "HP LaserJet", //Printer NAme
       blob: myReport, //The report blob object
    });
```

cloudQuery

Requests Data from a CloudGateway Enpoint

Parameters

```
{
    AppName: "MyCloudGateway", //The Name in the ApiKey List
    Endpoint: "LocalFirebird", // Configured Endpoint
    SqlStatementId: "fbGetAddress", //Optional statemnt id, depending on the Plugin
    Parameters: { //Optional Parameters
      "id": 1
    },

}
```

Example Usages

Query an address from a firebird database

```
   brixxApi.cloudQuery({
    AppName: "MyCloudGateway",
    Endpoint: "LocalFirebird",
    SqlStatementId: "fbGetAddress",
    Parameters: {
      "id": 1
    },

})
```

closeModal

If the current brixxbox app is startet in a modal window, that window will be closed.

Example Usages

```
    brixxApi.closeModal();
```

composeEmail

Shows the email compose Dialog, or sends an email without an interface popup.

Parameters

1. emailOptions - (optional) Json object with email parameters.
   - to - array of email addresses
   - cc - array of email addresses
   - bcc - array of email addresses
   - text - message text
   - subject - subject
   - from - single string with an email address
   - replyTo - single email, that will be used as to reply to
   - autoSend - if set to true, the mail will be send automatically. You will not see a dialog at all. (Default is false)
   - hideToEmail - if set to true. The recipients email address is hidden in the email dialog
   - attachmentIds - an array of attachment ids that will be sent as an attachment,
   - attachmentBlobs - an array of blobs objects that will be sent as an attachment, {name: myFileName, blob: myBlob}
   - recordId - RecordId if you would like to attach the email to an record. If empty, this is the current record Id (available from 18.11.2021)
   - appName - AppName if you would like to attach the email to an record. If empty, this is the current App Name (available from 18.11.2021)

Example Usages

Example 1

Shows the dialog without any predefined values

Shows the dialog text and to addresses

```
composeEmail({
   text: `This is the messsage text and this is a link: <a href='http://www.brixxbox.net' target='_blank'>www.brixxbox.net</a
   to:["john.doe@acme.com", "jane.doe@acme.com"],
   subject: "This is the mail subject",
   from: "noreply@acme.com"
});
```

Example 2

Sends the email without a dialog

```
composeEmail({

   text: "This is the messsage text",

   to:["john.doe@acme.com", "jane.doe@acme.com"],

   subject: "This is the mail subject",

   from: "noreply@acme.com",

   replyTo: "reply@acmy.com",

   autoSend: true //Set this to avoid the dialog

});
```

Example 3 - attachment blobs

```
composeEmail({
   text: "This is the messsage text",
   to:["john.doe@acme.com", "jane.doe@acme.com"],
   subject: "This is the mail subject",
   from: "noreply@acme.com",
   autoSend: true,
   attachmentBlobs: [
      {
         name: "Test1.pdf",
         blob: app.createReport("report")
      },
      {
         name: "Test2.pdf",
         blob: app.getAttachmentById(2)
      }
   ],
});
```

Example 4 - attachment ids

```
composeEmail({
   text: "This is the messsage text",
   to:["john.doe@acme.com", "jane.doe@acme.com"],
   subject: "This is the mail subject",
   from: "noreply@acme.com",
   autoSend: true,
   attachmentIds: [1, 2, 3]
});
```

createPublicAppUrl

Creates an url, that leads to a standalone app for pulic use. There is still a valid user logged in. The public user has to be configured in the settings as a "valid external user" for security reason

Parameters

1. options - Json object with the same properties as [createPublicAppUrl](/globalDoc/function_createPublicAppUrl) options, plus:
   - publicUser - the user email address of the brixxbox user, that will be used for the public app .

Example Usages

```
await brixxApi.createPublicAppUrl({appName: "survey", publicUser: "surveyuser@acme.com", id:1});
```

copyConfigRecordById

Creates a deep copy from a config record. Typical example is to create a copy of an "order" record, with a copy of all the "orderLines" from that source "order". brixxbox will use the "Cascade Copy" flag of a grid control, to decide if which referenced records will be copied.

Parameters

1. configName
2. recordId - the id of the source record
3. additionalValues - json object of values in the target record that should be modified. e.g. an "orderDate" field. Can be null, in this case you get a 1:1 copy except the record id.

Example Usages

Exact copy:

```
let newOrder = brixxApi.copyConfigRecordById("customerOrder", 1376);
```

Copy with new date for header data and manipulating two columns of line item data:

```
let newOrder = brixxApi.copyConfigRecordById("customerOrder", 1376, {
   cordOrderDate: new Date(),
   cordlnOrderQuantity: "1",
   cordlnDeliveredQuantity: null
});
```

createReport

Creates a report on the Server and retrieves it to the client. The only format, currently supported is pdf.

Parameters

1. reportControlId - the id of the reportControl in your app. This can be invisible, if you don't need a preview.
2. createOptions - JSON Object with options
   - print - (bool, default = false) if set to true, this will trigger the print dialog after the report is create.
   - archive - (bool, default = false) if set to true, the report is saved as an attachment to the current record,
   - saveFile - if set. the report will be downloaded with a generic file name (brixxbox-print.pdf), except, if saveFileName is set.#
   - saveFileName - if set. the report will be downloaded with that file name.
   - reportId - (optional) if defined, this will overruled the reportId given for the control in the config editor. This is not the controlId, but the id (name definition
   - cloudPrinter - (optional) the created report will be send to the cloud printer right after creation.

Like a on a dataRequest, all controls of the current app will bo automatically send as report parameters, plus the recordId of the current record as "id" have a control "id" in your app.

Example Usages

```
brixxApi.createReport("invoiceReport", {print: true}); //will create the invoiceReport as a pdf and prints it on the client si
```

```
brixxApi.createReport("invoiceReport", {
   print: true,
   archive: true,
   documentTypeId: 1,
   saveFileName: "myPrint.pdf"
});
//will create the invoiceReport as a pdf and prints it on the client side, it will also save the pdf as an attachment to the c
```

```
brixxApi.createReport("invoiceReport", {
   print: true,
   reportId: "invoiceReportSimple"
});
//In this case, we choose a different reportId
```

Create a report and send it to a cloud printer

```
brixxApi.createReport("invoiceReport", {
   cloudPrinter: "HP LaserJet",
});
```

deleteAttachment

Deletes an attachment and its link to a record.

Parameters

- attachmentId - the id of the attachment you want to delete

Example Usages

```
    brixxApi.deleteAttachment(42);
```

deleteConfigRecordById

Deletes a record of a given config

Parameters

1. configName - the name of the config
2. id - the record id you want to delete

Example Usages

```
    brixxApi.deleteConfigRecordById("address", 123);
```

deleteRecord

Deletes a record from the database.

Parameters

1. options - json object
    - id - the record id. Optional. By default, this is the current record id. (The current record id can be accessed here: brixxApi.actualRecordId)
    - noConfirmMessage - Optional. By default, this is false. If set to true, you will not get a confirm MessageBox. the record will be deleted.

Example Usages

1. Simple

```
brixxApi.deleteRecord(); //deletes the current displayed record
brixxApi.deleteRecord({id: brixxApi.actualRecordId}); //deletes the current displayed record
```

2. Simple - No Confirmation

```
brixxApi.deleteRecord({noConfirmMessage: true}); //deletes the current displayed record without confirmation
```

3. Without displaying the record

```
let myRecord = await brixxApi.laodRecord("myKeyControl"); loads a record into myRecord
brixxApi.deleteRecord({id: myRecord.id}); //deletes the loadedRecord
```

displayRecord

Displays a record in an brixxbox app. To get a record, you could use e.g. loadRecord

Parameters

1. the record.
2. the current control. This control is excluded from the discard changes check, that occurs before the record is displayed

Example Usages

Simple (pay attention to the **await** keyword)

```
await brixxApi.displayRecord(myRecord, "myKeyControlId");
```

downloadAttachments

Downloads a list of attachment ids as a zip file.

Parameters

1. downloadOptions - JSON object
    - requestedIds - list of ids to download
    - fileName - (optional) name of the downloaded file

Example Usages

```
brixxApi.downloadAttachments({
   requestedIds:[196, 197, 200]
});
```

```
brixxApi.downloadAttachments({
   requestedIds:[196, 197, 200],
   fileName: "myDownload.zip"
});
```

disableGridHyperLinks

Disables (or enables) the hyperlinks in grid cells, automatically generated for combo box controls. This function also refreshes the grid.

Parameters

1. controlId - The id of the grid control
2. disable - true, if you want to disable the hyperlinks, false if you want to enable them again.

Example Usages

```
    brixxApi.disableGridHyperLinks("myGrid", true); //disable links
    brixxApi.disableGridHyperLinks("myGrid", false); //enable links again
```

enableNotifications

This function is used to disable or enable app messages, that are displayed automatically, like the "record Saved" message, after a successful save operation.

Parameters

1. param - The type of message. The following messages types are supported
   - "save" - The save Message

Example Usages

```
    brixxApi.enableNotifications("save", false); //Disable the save message

    brixxApi.enableNotifications("save", true); //Enable the save message (This is the default)
```

enableValidator

Enables or disables field validator.

Parameters

1. controlId - The control id of the field
2. validatorName - the name of the validator
3. enable - enable (true=default) or disable (false) the validator

Example Usages

```
brixxApi.enableValidator("firstName", "notEmpty", false); //disables the validation of notEmpty for the control firstName
```

excludeFromReadonly

It provides the functionality of excluding controls from being affected by the readOnlyMode.

In our doc for **readOnlyMode**, we have seen how we can use it to disable all app controls or some of them. For that we will always have to make a call to **readOnlyMode** function, if we want anything to change. **excludeFromReadonly** function provides the functionality of excluding controls from being affected by the readOnlyMode. For example, we can set this function in appStart event then it will effect all **readOnlyMode** function calls after that. It means if we have set some controls to be excluded for readONly Mode function at app start, they will always be excluded from it influence during app life time and we do not have to make calls to this function again and again. If user wants to update **excludeFromReadonly** controls list then a new call to **excludeFromReadonly** will overwrite the previous setting.

Exaample

```
brixxApi.excludeFromReadonly(["myFirstControl", "mySecondControl"]);
```

Parameters

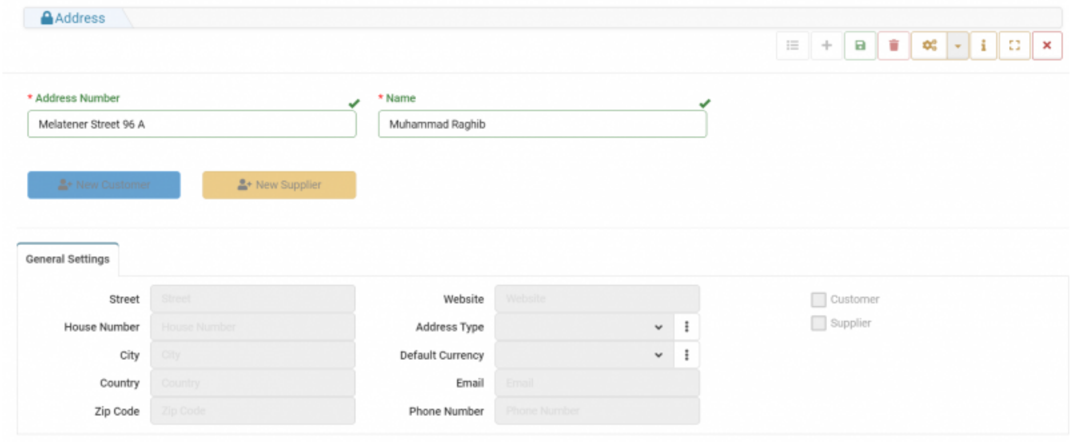
1. Control array - It is a json array of control ids.

Example Usages

As we have seen in **ReadOnlyMode** the example of address app and how we disabled every control using it without options. Now let say we want "Address Number" and "Name" controls to stay enabled. For this purpose, we will need to add these controls to our **excludeFromReadOnly** function and put this code in appStart event.

```
brixxApi.excludeFromReadonly(["adrNumber", "adrName"]);
```

As we can see in snapshot below Address number and name controls are editable after the app is started. We can use this functionality with any number of apps.



* Address Number: Melatener Street 96 A
* Name: Muhammad Raghib

Street	Street	Website	Website	☐ Customer
House Number	House Number	Address Type		☐ Supplier
City	City	Default Currency		
Country	Country	Email	Email	
Zip Code	Zip Code	Phone Number	Phone Number	

executeStoredProcedure

Executes a stored procedure.

Parameters

1. procedureName - The name of the stored procedure
2. procedure parameters - If you need to add parameters, you can use this json object to set them
3. queryOptions - (optional) a json object with options for the request
   - timeout - (optional) timeout for the SQL Request. Default is 30 seconds
   - connectionKey - (optional) a key to a custom settings entry with the connection string to an external MSSQL database

Example Usages

1. Simple (pay attention to the **await** keyword)

```
await brixxApi.executeStoredProcedure("procedureName");
```

2. With parameters (pay attention to the **await** keyword)

```
let parameterJson = {
   myCustomSqlParameter: 12345
};
await brixxApi.executeStoredProcedure("procedureName", parameterJson);
```

3. Short version (pay attention to the **await** keyword)

```
await brixxApi.executeStoredProcedure("procedureName", {
   myCustomSqlParameter: 12345
});
```

4. With Timeout

```
await brixxApi.executeStoredProcedure("procedureName", {
   myCustomSqlParameter: 12345
},{
   timeout: 45
});
```

getAttachmentId

Returns the id of an attachment of the current record by looking for the given file name.

Parameters

fileName - the name of the file

Example Usages

```
    brixxApi.getAttachmentId("Invoice1.pdf")
```

getAttachmentById

Gets and attachment by its unique id.

Parameters

id - id of the attachment

Example Usages

```
brixxApi.getAttachmentById(1234);
```

```
let allAttachmentsForThisRecord = brixxApi.getAttachmentsForCurrentRecord();
let invoices = allAttachmentsForThisRecord.filter(singleAttachment => {return singleAttachment.documentTypeId == 2}); //We ass
let attachmentBlob = brixxApi.getAttachmentById(invoices[0].id);
brixxApi.printBlob(attachmentBlob)
```

getAttachmentByFileName

Retrieves the blob of an attachment from the attachments of the current record

Parameters

fileName - the file name you are looking for

Example Usages

```
    brixxApi.getAttachmentByFileName("Invoice1.pdf")
```

getAttachmentsForCurrentRecord

Returns a list of all the attachments of the current record. The list is ordereb by id desc. so the newest documents will iterating over the array.

Example Usages

Example 1

```
console.log(brixxApi.getAttachmentsForCurrentRecord());
```

Example 2 Filtering

```
let allAttachmentsForThisRecord = brixxApi.getAttachmentsForCurrentRecord();
let invoices = allAttachmentsForThisRecord.filter(singleAttachment => {return singleAttachment.documentTypeId == 2}); //We ass
let attachmentBlob = brixxApi.getAttachmentById(invoices[0].id);
brixxApi.printBlob(attachmentBlob)
```

getBrowserToken

Gets a GUID Token, that identifies the browser. This token will **NOT** be different, when you close all browser tabs and start the brixxbox again. But it will be different, when you delete the local storage for that url. You could use this to identify a shopping cart, for example.

Example Usages

```
    let myToken = brixxApi.getBrowserToken();
```

brixxApi.getSessionToken

GUID stands for Global Unique Identifier. A GUID is a 128-bit integer (16 bytes) that you can use for authentication purposes. There are many types of GUID tokens, one of them is browser token. We can use this token to identify a user for same browser because his token will **NOT** be different, when you close all browser tabs and start the Brixxbox again. But it will be different, when you delete the local storage for that url. This function can be used to get GUID for browser. It can then be used to keep user session open. Another usecase is that it could be used to identify a shopping cart. An important aspect here to note is that when when you close all browser tabs and start the Brixxbox again, this token will not be different.

Example Usages

An example user is that user can get access tokens using this **getBrowserToken** functions from two apps and compare them to confirm user identity i.e after login this can be used as authenticate the user between different browser sessions.

getControlUnmodifiedValue

Gets the unmodified value of the control.

Example Usages

```
    let unmodifiedCompareValue = brixxApi.getControlUnmodifiedValue("myControlId");
```

getCustomMessage

Returns a custom message from the message manager. It replaces possibly passed parameters in the message. The target language can also be set.

Example Usages

```
    let msg = brixxApi.getCustomMessage("messageName", {"param1": "text1", "param2": "text2"}, "en-Us");
    brixxApi.setFieldValue("controlId", msg);
```

getCustomSetting

Retrieve a value for a specific custom setting.

Parameters

1. settingName - the key for the setting

Example Usages

```
let settingValue = brixxApi.getCustomSetting("groupOne.setting1");
console.log(settingValue);
```

getCalcDateTime

Gets a DateTime as a compareable number with a given accuracy.

Parameters

1. dateTime value or controlId
2. accuracy - "seconds" (default), "minutes" or "hours"

Example Usages

```
brixxApi.getCalcDateTime(myDateTimeControl);
brixxApi.getCalcDateTime(myDateTimeControl, "minutes");
brixxApi.getCalcDateTime(myDateTimeControl, "hours");
```

Example 2 calculation

```
let durationInSeconds = brixxApi.getCalcDateTime(myDateTimeEnd) - brixxApi.getCalcDateTime(myDateTimeStart);
let durationInMinutes = brixxApi.getCalcDateTime(myDateTimeEnd, "minutes") - brixxApi.getCalcDateTime(myDateTimeStart, "minute
let durationInHours = brixxApi.getCalcDateTime(myDateTimeEnd, "hours") - brixxApi.getCalcDateTime(myDateTimeStart, "hours");
```

getHtmlElement

Gets the html element for this brixxbox control.

Note: Usually it is not necessary to get the Html element itself. Make sure there is no better way to do the things you want to do by using the brixxApi functions!

Example Usages

```
    let myField = brixxApi.getHtmlElement("myFieldId");
    myField.disabled = true; // just for demonstration. Please use brixxApi.setFieldEnable("myFieldId", false) instead
```

getDate

As the name suggests, this functions is to get date. This function when used without any arguments returns a date in this format 'yyyy-mm-dd'.

It is very important to be cautious because different date formats can lead into worng results when in comparison.

Parameters

1. param - (optional). If empty it will return todays date. User is also allowed to provide a date or moment object of javascript.

Example Usages

Following examples shows all three usages of getDate function. Lets suppose we want to know the new order placement date which is today then we can use 1st or second example. We also suppose that our order delivery date is after one month then we can use the third example and use this date as a delivery date.

```
brixxApi.getDate();
brixxApi.getDate(new Date());
brixxApi.getDate(new moment().add(1, "month"));
```

getCalcDate

Converts a date to the total days since 1.1.1970 to make it easy comparable. You can use Date, or DateTime as input and even if two datetime values do have different times.

Parameters

inputDate - This can be a date or datetime in any format or a controlId to a date or dateTime control

Example Usages

```
let calcDate1 = brixxApi.getCalcDate(brixxApi.getFieldValue("myDateTime1"));
let calcDate2 = brixxApi.getCalcDate(brixxApi.getFieldValue("myDate2"));
if(calcDate1 === calcDate2){
   //Do something
}
```

Example Results

```
brixxApi.getCalcDate(new Date('1995-12-17 03:24:00')) //9481 (days since 1.1.1970)
brixxApi.getCalcDate('1995-12-17 03:24:00')           //9481
brixxApi.getCalcDate('1995-12-17 12:44:30')           //9481
brixxApi.getCalcDate(myControlId)                     //?
```

getFieldValue

Gets a value to a specific field

Parameters

1. controlId => String with the control id. Some controls (like Comboboxes) do have sub information. in that case the controlId is combined with a "."

Example Usages

```
let myText = "Hello World!";
brixxApi.setFieldValue("myControlId", myText);
value = brixxApi.getFieldValue('fieldId');
```

```
let myFieldId = 'fieldId';
let value = brixxApi.getFieldValue(myFieldId);
```

```
let itemId = brixxApi.getFieldValue('itemCombobox'); //like any other control
let taxKeyId = brixxApi.getFieldValue('itemCombobox.itmTaxKeyId'); //get the itmTaxKeyId value of the selected entry in acombo
let itemName = brixxApi.getFieldValue('itemCombobox.itmName');
```

```
let taxKeyId = brixxApi.getFieldValue('itemList').itmTaxKeyId; //get the itmTaxKeyId value of the selected entry in a grid
let allRowsFromMyGrid = brixxApi.getFieldValue('itemList.allRows'); //get an array of all rows in a grid
```

```
let selectedRowsFromMyGrid = brixxApi.getFieldValue('itemList.selectedRows'); //get an array of all selected rows in a grid
let unselectedRowsFromMyGrid = brixxApi.getFieldValue('itemList.unselectedRows'); //get an array of all unselected rows in a g
let myRowJsonObject = brixxApi.getFieldValue('itemList.clickedRow'); //get the clicked row as a json object (only valid in onR
let myCellValue = brixxApi.getFieldValue('itemList.clickedCell'); //get the clicked cells formatted value (only valid in onCel
let myCellId = brixxApi.getFieldValue('itemList.clickedCellId'); //get the clicked cells id (only valid in onCellClick events)
```

```
for(let i = 0; i < allRowsFromMyGrid.length; i++){
console.log(allRowsFromMyGrid[i].itmTaxKeyId);
}
```

getSessionToken

GUID stands for Global Unique Identifier. A GUID is a 128-bit integer (16 bytes) that you can use for authentication purposes. We use this token to identify a user for same browser session. This function can be used to get GUID for current user sessions. It can then be used to authenticate user between different apps. Another usecase is that it could be used to identify a shopping cart. An important aspect here to note is that when when you close all browser tabs and start the Brixxbox again, this token will be different.

Example Usages

An example user is that user can get access tokens using this **getSessionToken** functions from two apps and compare them to confirm user identity i.e after login this can be used as authenticate the user.

```
    let myToken = brixxApi.getSessionToken();
```

getUserClaim

Brixxapi allows user to set a claim value. This claim Value gets added when user gets logged in. It is used to identify users claim. For example, to che
whether a user is someone who he claims to be or not. By using this function, we can get userClaim and make useful comparisons in our apps.

Parameters

1. claimName - This is the only paramter this function takes. It is the claiName which should be in the form of string.. Brixxapi allows user to add cla
   name with or with out "claim_" prefix.

Example

```
Let suppose we have a claimName parameter "EFitDealer" then Brixxapi will consider both "EFitDealer" and "claim_EFitDealer" as sa
```

Example Usages

We can use "getUserClaim" function to get user's claim, a way to check user identity. As a parameter, we need to provide the claimName to this funct
We can provide only the claimName or the claimName plus "claim_" as prefix. Lets see both of them in action. We have a claimName "EfitDealerNur
now we want to get userClaim:

Lets first do it without prefix "claim_".

```
    let dealerNumber = brixxApi.getUserClaim("EfitDealerNumber");
    console.log("The dealer Number is: " + dealerNumber);
```

It is ame as the previous example but now we use prefix.

```
    let dealerNumber = brixxApi.getUserClaim("claim_EfitDealerNumber");
    console.log("The dealer Number is: " + dealerNumber);
```

An important thing to notice here is that it is required by Brixxbox that for using SQL Parameters for claims must have to start with the "claim_" prefix.

For example, Userclaims can also be used as SQL parameters:

```
    select @claim_EfitDealerNumber
```

getConfigRecordAttachmentIdByFileName

Returns an id of an attachment of a record in a config.

Parameters

1. appName - the name of the config with the attachment
2. recordId - the id of the record with the attachment
3. fileName - the name of the requested attachment

Example Usages

```
brixxApi.getConfigRecordAttachmentIdByFileName("customerOrder", 1234, "invoice.pdf");
```

getSignatureImageBlob

Returns the signature of a signature control as a png (default) or svg image blob.

Parameters

1. controlId - The id of the signature control
2. type - optional, if not set, the function returns a png image. If set to "svg", the function returns a svg vector image.

Example Usages

```
let myBlob = brixxApi.getSignatureImageBlob("mySignature"); //gets the blob as PNG
```

```
let myBlob = brixxApi.getSignatureImageBlob("mySignature", "svg"); //gets the blob as SVG
```

```
brixxApi.printBlob(brixxApi.getSignatureImageBlob("mySignature"));
```

getGeoLocation

Returns the geo location and additional information like speed and altitude if available. Be aware, that this is an asynchronous function and you have to use await.

The Result looks like this:

```
coords
   accuracy: 33
   altitude: null
   altitudeAccuracy: null
   heading: null
   latitude: 51.073834399999996
   longitude: 6.0768204
   speed: null
timestamp: 1557477072518
```

Example Usages

```
let myGeoLocation = await brixxApi.getGeoLocation();
console.log(myGeoLocation.coords.latitude);
console.log(myGeoLocation.coords.longitude);
```

To Visualize a geo location in google maps, see this documentation(http://www.joerg-buchwitz.de/temp/googlemapssyntax.htm) for possible url parameters, You could use the following commands in an event:

```
let myLocation = await brixxApi.getGeoLocation();
window.open("http://maps.google.de/maps?q=" + myLocation.coords.latitude + "," + myLocation.coords.longitude,'_blank');
```

globalSearch

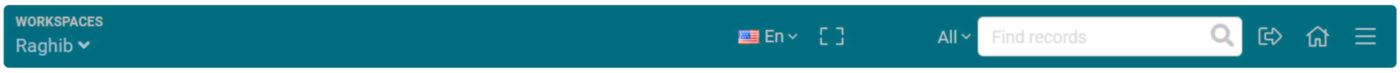
This functions is used to trigger global search of the whole workspace. It only takes one parameter which is a search keyword. This function is directly used in the each Brixxbox workspace. User can find it in top right corner of each workspace page. It is a textbox accompanied by search button. User needs to enter a string in the global search field at the top of the screen and it will trigger the search function. It can be useful if you want to search for a scan result or anything within your workspace in seconds.

Parameters

1. searchTerm - It is of type string.

Example Usages

User can find this button on top right corner. It looks like this.



User can enter any keyword and it with return thespecific results. User can also use this function in their apps. For this user need to provide a search keyword and can use it to retreive data from the global workspace. If user want to search "address" the code will look like this

```
brixxApi.globalSearch("address"); //triggers a search for "address" in global address space.
```

initAllControls

Initializes all controls within a brixxbox App to their empty states.

Example Usages

```
    brixxApi.initAllControls();
```

initControl

Initializes a control to its empty state

Example Usages

```
brixxApi.initControl();
```

isModal

Determens if the current brixxbox app is started in a modal window.

Example Usages

Simple

```
    if(brixxApi.isModal()){
       alert("the brixxbox app is modal!");
    }
```

isUserInRole

All users in any Brixxbox workspace shoyuld have been assigned a specific user role. isUserInRole takes a string argument role. On the basis of this role parameter, it determines whether a logged user

```
    brixxApi.isUserInRole("Management");
```

Parameters

1. role - It is a string parameter. It is the name of logged in user role which needs to be checked.

Example Usage

We have seen address app in the docs of **ReadOnlyMode** function. Now we will build upon that functionality. In it we have used **readOnlyMode** function to disable all ap controls. Now we want tht if user is of admin role then all the controls should be enabled and disabled otherwise. For this we need to update our custom code. It should look like this.

```
if(brixxApi.isUserInRole("Admin")){
    brixxApi.readOnlyMode(false);
    console.log("raghib")
}
else{
    brixxApi.readOnlyMode();
}
```

Now If user 'raghib' has an admin role assigned to him then address app will be enabled for him and for all other non admin users. It will be disabled.

isRowSelected

Checks if a row in a grid is selected or not.

Parameters

1. controlId - The id of the grid
2. line - This can be either a whole line from a "myGrid.allRows" call or a single id from a record.

Example Usages

```
let result = brixxApi.isRowSelected("myGrid", myLine)
```

Used in a loop

```
let allRows = app.getFieldValue("myGrid.allRows");
for(let i = 0; i < allRows.length; i++){
   if(brixxApi.isRowSelected("myGrid", allRows[i])){
      console.log("This is selected: " + line.id);
   }
}
```

Used with an id

```
let result = brixxApi.isRowSelected("myGrid", 123); //Checks if the line with the id 123 is selected.
```

inviteUser

inviteUser(string email)

This function takes a string as a parameter. This string should be an email address of a new user to whom admin of current workspace wants to invite. This function provides functionality of sending an email to a new user.

Example Usage

```
var result = brixxApi.inviteUser("john.doe@acme.com");
```

In order to learn more about this function click here.

localValue

Stores and retrieves local values, like settings or something, that you want to have persistent on this specific browser. The value will be saved only for the browser you are currently working on. The value will be save in the scope of the current workspace you are working in.

Parameters

1. key - Key name
2. value - (optional) value you want to save. If this is empty, brixxbox will return the value, you saved before.

This is **not** an asynchronous operation. **No await needed!**

Example Usages

```
    brixxApi.localValue("myKey", "Hello World"); //Save the value "Hello World" under the key "myKey"
    let x = brixxApi.localValue("myKey"); //Retrieve the Key "myKey". This will result in "Hello World"

    console.log(x)
```

loadAndDisplayRecord

Loads a record, based on the value in the control, given as a parameter in the first argument and displays it. This function just combines loadRecord and displayRecord

Example Usages

Simple (pay attention to the **await** keyword)

```
brixxApi.loadAndDisplayRecord(keyFieldID);
```

```
await brixxApi.loadAndDisplayRecord(keyFieldID);
```

loadAndDisplayRecordById

Loads a record, based on the value in the control, given as a parameter in the first argument and displays it. This function just combines loadRecordById and displayRecord displayRecord

Example Usages

Simple (pay attention to the **await** keyword)

```
let recordId = 123;
brixxApi.loadAndDisplayRecordById(recordId);

await brixxApi.loadAndDisplayRecordById(recordId);
```

loadConfigRecordById

Loads a record of a specific app config from brixxbox server and returns a promise. This is a async function. You have to await the result, if you want to use it.

Parameters

1. configName - the name of the app
2. id - recordId to load

Result

The result is a brixxbox record object. All fieldvalues of that record are stored in data as key value pairs. (e.g. myRecord.data.addressName). The id of the record can be found in myRecord.id.

Example Usages

Simple (pay attention to the **[await](/globalDoc/glossary_await)** keyword)

```
    let myAddress= await brixxApi.loadConfigRecordById("address", 15);
    let addressName = myAddress.data.name;
    let id = myAddress.is;
```

loadRecord

Loads a record, based on the current config, from brixxbox server and returns a promise. This is a async function. You have to await the result, if you want to use it. This function will not display the record, it will just load and return it. To display the record, use the function result with displayRecord.

Parameters

The first parameter is a control with the key value, that will be used to find the record (like an address number)

Example Usages

Simple (pay attention to the **await** keyword)

```
    let myRecord = await brixxApi.loadRecord("myKeyControl");
```

loadRecordById

Loads a record from brixxbox server and returns a promise. This is a async function. You have to await the result, if you want to use it. This function will not display the record, it will just load and return it. To display the record, use the function result with displayRecord.

Parameters

The first parameter is a record id

Example Usages

Simple (pay attention to the **await** keyword)

```
    let myRecordId = 123;
    let myRecord = await brixxApi.loadRecordById(myRecordId);
```

logAdd

The brixxbox offers a client-side event log. Events like save/delete record, data validation messages, internal events etc. will be logged. In addition, the log can be accessed through brixxApi. The log view is located in the side bar. It will be shown in an extra tab page (only if data is available). Per default only entries for the current brixxApi Object (and its children) will be visible.

Parameters

Parameters can be passed as a json object.

1. text: the actual log text. If this value is missing no entry will be generated.
2. context: This should be set to a plausible group-criteria (e.g. “save” for record saved. “validate” for data validation). This value can also be used to remove the complete group from the log (see example).
3. status: Status can have the following values “info”, “success”, “warning”, “danger” or “error”. The log entry will get its color set accordingly. “danger” and “error” are interchangeable.
4. info: Additional info can be set. If empty its value will reflect the app title of the brixxApi object.
5. recordId: If not passed as a parameter the value will be set to record-ID of the brixxApi object (brixxApi.record.id).

Example Usages

```
brixxApi.logAdd({context: "GroupCriteria", status: "info", text: "Lorem ipsum dolor sit amet, consectetur adipisici elit"})
```

To remove an existing context from the log for the current brixxApi object **logReset** is available:

```
brixxApi.logReset("GroupCriteria")
```

logout

Logs the user out. The user has to go to the login screen again to get access.

Example Usages

```
brixxApi.logout();
```

markAllControlsAsUnModified

Removes all modification flags from the controls. After this call, the user will not get a "discard changes" message until another control is modified.

Example Usages

```
    let value = brixxApi.markAllControlsAsUnModified();
```

messageBox

Displays a modal messagebox with buttons. The function will return when a button is clicked and return the value, if set, otherwise the title of the clicked button

Parameters

messageBoxOptions - A JSON object with messageBox parameters

Example Usages

```
let msgBoxResult = await brixxApi.messageBox({
   title: "sampel title",
   text: "sampel text",
   buttons: [
      {
          title: "Yes",
          value: "1"
      },
      {
          title: "No",
          value: "0"
      },
      {
          title: "Cancel",
          value: "canceled"
      },
      {
          title: "Demo" // when Demo is clicked it will return "Demo" because the button does not provide a value option.
      }
]});

console.log(msgBoxResult); //This will be either "1", "0", "canceled" or "Demo"
if(msgBoxResult==1){
   //enter code
}
```

newGridEntry

Creates a new grid line for a data grid.

Parameters

The first parameter is a grid control

Example Usages

Simple (pay attention to the **await** keyword)

```
    await brixxApi.newGridEntry("myGridControl");
```

newRecord

Initializes the app, so that you can enter a new record

Example Usages

Simple (pay attention to the **await** keyword)

```
    await brixxApi.newRecord();
```

printBlob

Prints a document that is a blob. You might get a blob by requesting an attachmen

Parameters

1. Print Document

Example Usages

```
brixxApi.printBlob(brixxApi.getAttachmentByFileName("Invoice1.pdf"));
```

print

Prints a report

Parameters

1. controlId - ControlId of the report control, that you want to print

Example Usages

```
    brixxApi.print(invoiceReport);
```

readOnlyMode

It can set the whole brixxbox app or some controls depending upon the use case which is required, in a readonly mode. All Input fields, save- new- and delete toolbar buttons are disabled.

In Brixxbox, we have certain types of users like system users those who build the system, users those who will use the system, and admins, they will manage the systems etc. Each user should have different access to the system. It means all acess rights should only be given to admin users. Lets suppose we have an orders app and recorded orders. System users should only be allowed to fetch and check orders while on the other hand admins shouls also be allowed to edit orders and make changes to them. This is this brixxApi **readOnlyMode** function comes into play.

Example

This code sets the whole app to read only mode. All the input controls and even the toolbar button get disabled.

Parameters

readOnlyMode function can be used with or without options. Following are the optional parameters:

1. readOnly - (bool, default = true) This indicates if the readOnly mode is enabled (true) or disabled (false). If it is set to true or without options, it disables the whole app. If set to false, the app starts to behave normally. Every control and each toolbar button is enabled.
2. options - It is a JSON object which can be passed as an additional options.
   - exclude - It is an array of controlIds that should be excluded from the disable functionality of readOnlyMode function.

Example Usages

We can use this "readOnlyMode" function with or without options. Lets see both of them in action.

Without options

To disable each and every control and toolbar buttons use this function without any options like

```
brixxApi.readOnlyMode(); // This will set app in readOnly Mode.
brixxApi.readOnlyMode(true); // This will also set complete app in readOnly Mode.
```

In order to enable all the controls, set the **readOnlyMode** function with false value of readOnly parameter. It will enable back all the controls and the app will start behaving normally.

```
brixxApi.readOnlyMode(false); // sets back to normal Mode.
```

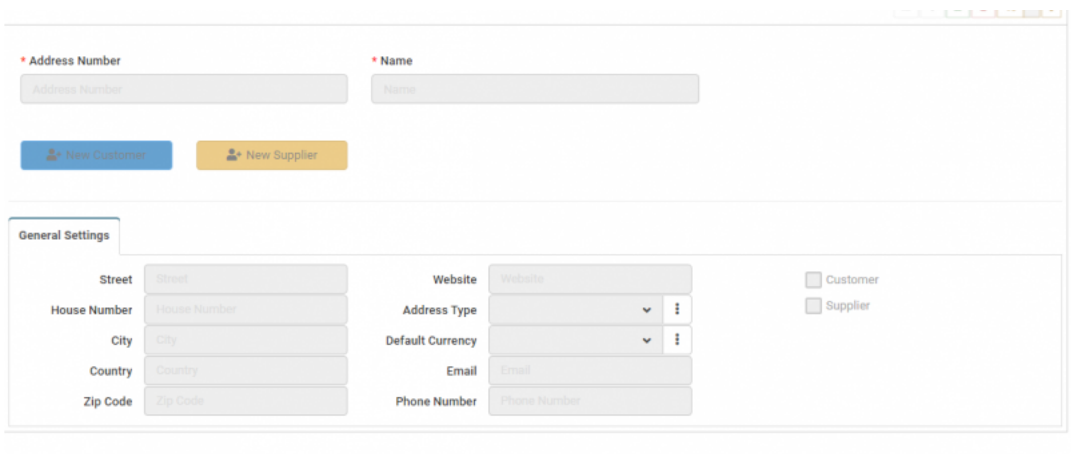
With options

Now if we dont want to use default functionality of disabling or enabling the whole app, we can use**readOnlyMode** options to enable or disable some of the controls. Lets say we want to disable 2 controls from an app then we have to pass the name sof these controls in exclude option array like

```
brixxApi.readOnlyMode(true, {
   exclude: ["myFrstControl", "mySecondControl"]
});
```

Example in Action

Lets say we have address app and we want to disable all of it. It can be done in different ways. For a moment, lets stick to "appInitialized" event. Add this function as a custom code of "appInitialized" event and save the app. It will look like this


* Address Number
* Name

New Customer
New Supplier

General Settings

Street		Website		Customer
House Number		Address Type		Supplier
City		Default Currency		
Country		Email		
Zip Code		Phone Number		

As we can see each control as well as app toolbar is also disabled.

refresh

Reloads the content of a single control (like combobox or grid) or of all such controls in an brixxbox application.

Parameters

Optional. A control, if you want to refresh just this control, or undefined to refresh all controls within an application.

Example Usages

1. Simple (pay attention to the **[await](/globalDoc/glossary_await)** keyword)

```
    await brixxApi.refresh("myKeyControl");
    await brixxApi.refresh();
```

refreshAttachments

Refreshes the attachment grid in the sidebar if it is visible

Example Usages

```
brixxApi.refreshAttachments();
```

refreshDataSource

Refreshes the content of a single control based on its DataSource (not the SubDataSource!). If outside parameters have changed, you can force the brixxbox to reload the content of that control.

Parameters

1. controlId - (optional) id of the control, that you want to reload. If this is empty, all controls with datasources are refreshed.

Example Usages

```
    brixxApi.refreshDataSource("myControlId"); //Refreshes just "myControlId"
```

```
    brixxApi.refreshDataSource(); //Refreshes all Controls with a DataSource in the app
```

refreshBadges

Refreshes all Badges of an app at once.

Example Usages

```
    brixxApi.refreshBadges();
```

reInitValidation

Resets all validators of an app to their default state. So you don't have to bother with enableValidator to reset them or what the default state is.

Example Usages

```
    brixxApi.reInitValidation();
```

replaceText

Replaces a text with the current values of the app.

Parameters

1. text - the text with parameters
2. additionalReplacings - json object with key value pairs

Example Usages

This example uses the controls of the app to replace the values.

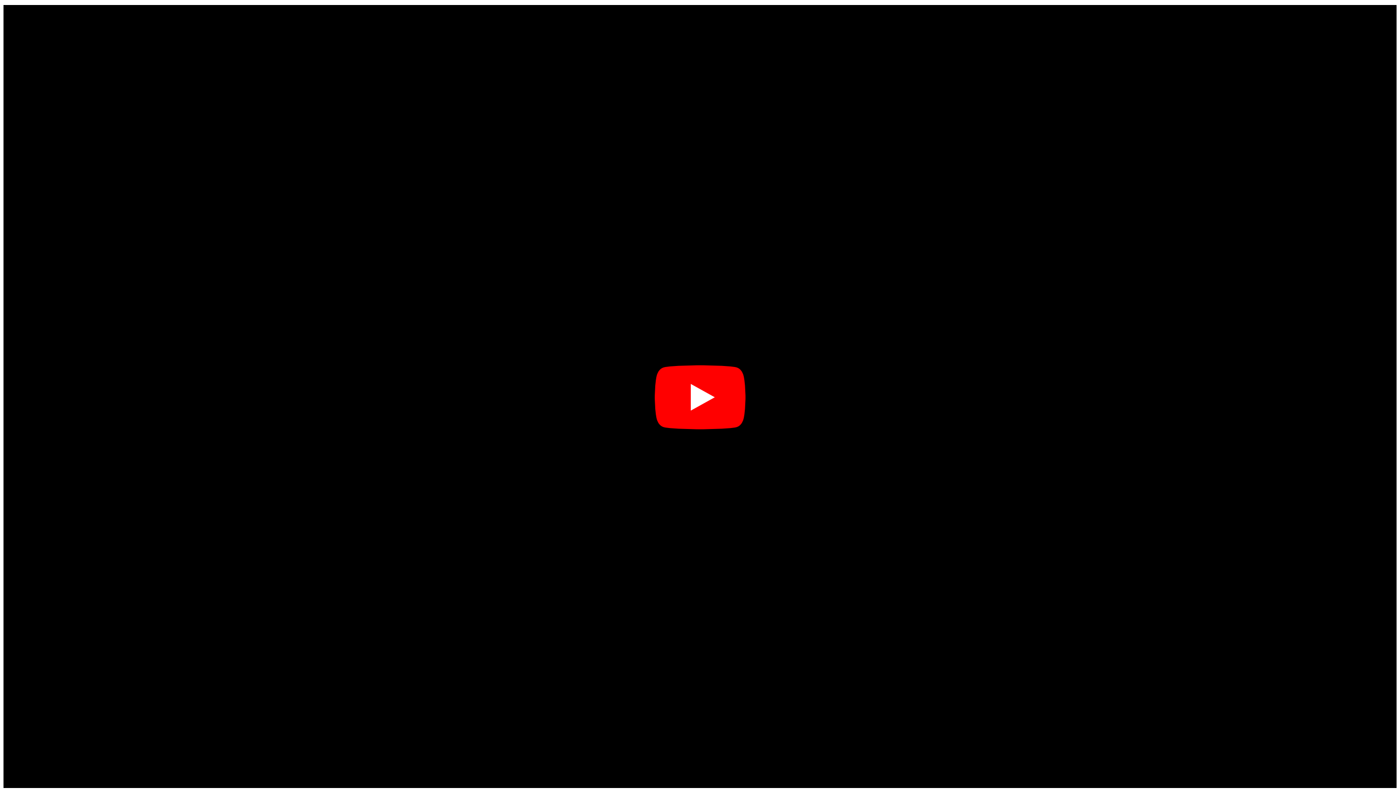
```
let newText = brixxApi.replaceText("Hello @id");
```

This example uses the controls of the app to replace the values.

```
let newText = brixxApi.replaceText("Hello @myCustomParam", {
 myCustomParam: "world"
});
```

serverFunction

Calls a server function and returns the result



Parameters

- functionName - the configured name of the function in App_ServerSideFunction
- functionsParameters - (optional) json object with function parameters. The will be available in the function under body.parameters
- options - (optional) See examples. You can use the options to get the payload of the request or a postman example, instead of calling the function. Can be used to debug a function on your local machine

Example Usages

To call a function and get the response object:

```
let functionResponse = brixxApi.serverFunction("myFunction", {
    name: "Hallo"
});
console.log(functionResponse.functionResult);
```

To get an example payload based on the current app state:

```
let functionResponse = brixxApi.serverFunction("myFunction", {
    name: "Hallo"
},{
    payload: true
});
console.log(functionResponse.functionResult);
```

To get an example postman collection including the payload of the current app:

```
let functionResponse = brixxApi.serverFunction("myFunction", {
   name: "Hallo"

},{
   postman: true
});
console.log(functionResponse.functionResult);
```

To define your own server functions please check Server Api Reference

setControlUnmodifiedValue

Sets the value, that is used to compare to the actual control value. If both are not equal, the field will be marked as changed.

1. Simple Example Usages

```
brixxApi.setControlUnmodifiedValue("myControlId", "unmodified Value");
brixxApi.setControlUnmodifiedValue("myControlId", 1); // sets the value to 1.
```

2. Simple Example Usages with variables

```
let valueToCompare = "Hello Wolrd!";
brixxApi.setControlUnmodifiedValue("myControlId", valueToCompare);
```

setFieldValue

brixxApi function to set a value to a field

Parameters

1. controlId => Id of the control
2. value => The value, you want to set. This could be a number, a date, a string (depending on the control type of course). setFieldValue will do all t

Example Usages

1. Simple textbox

```
brixxApi.setFieldValue("myControlId", "Hello World!");
```

2. With a variable

```
let myText = "Hello World!";
brixxApi.setFieldValue("myControlId", myText);
```

3. Setting a date

```
brixxApi.setFieldValue("myDateControl", "2019-01-31"); //This is the format, you would get by getFieldValue
brixxApi.setFieldValue("myDateControl", new Date()); //This will set todays date.
brixxApi.setFieldValue("myDateControl", new moment().add(1, "month")); // you can use moment to set calculated dates. See http
```

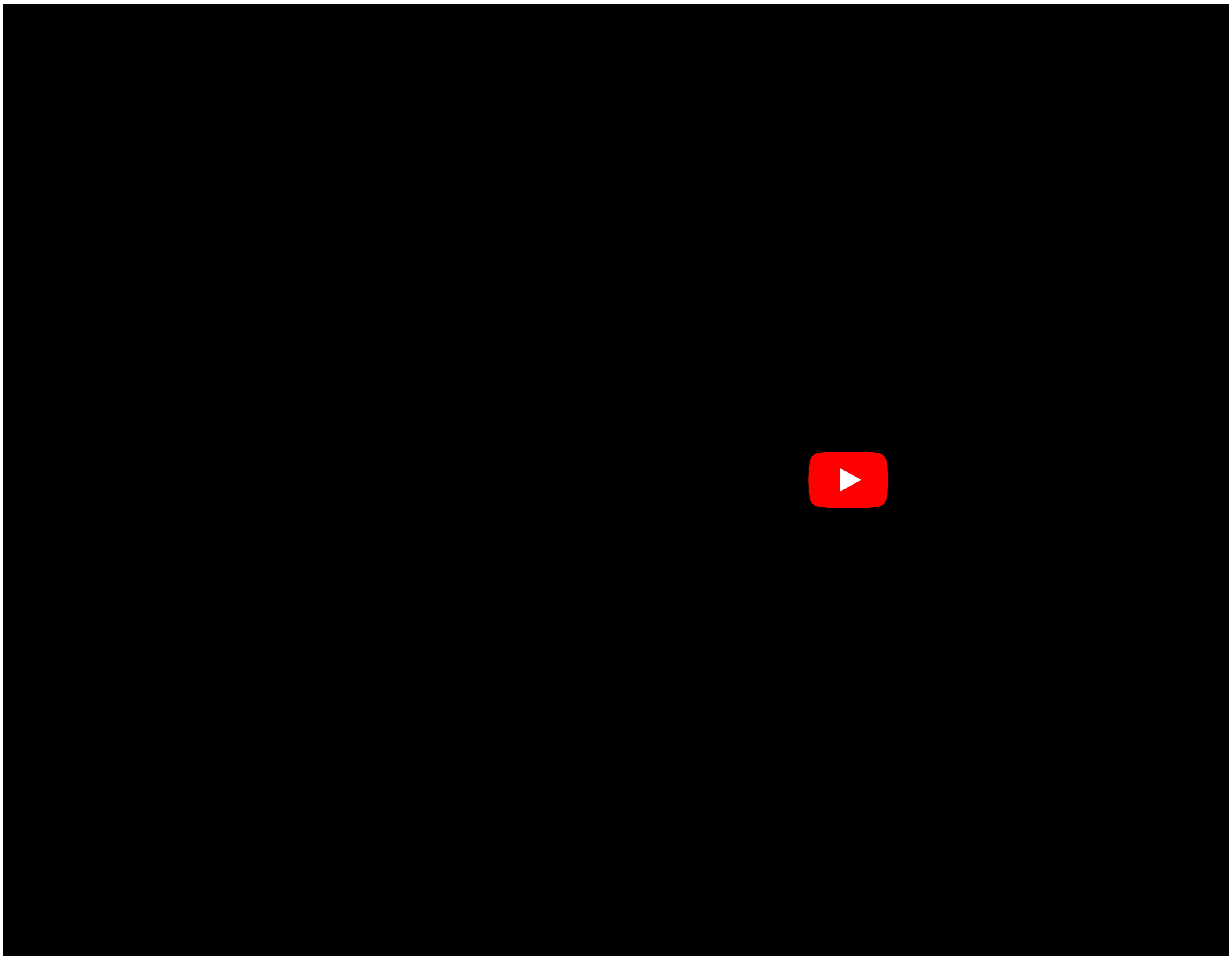
4. Setting a date to a calendar control scrolls to that date.

```
brixxApi.setFieldValue("myCalendarControl", "2019-01-31");
```

5. The Calendar supports subcontrols to set. You can add resource(s) to the calendar for example:

```
//resources will delete all and set a new array of resources
brixxApi.setFieldValue("myCalendarControl.resources", [
{
    id: 1,
    title: "Room A",
},
{
    id: 1,
    title: "Room B",
}]);

//resource will add a single resource
brixxApi.setFieldValue("myCalendarControl.resource", {
    id: 1,
    title: "Room A",
});
```

Demo



setLabelText

Sets the label text for a control at runtime.

Parameters

1. controlId => Id of the control
2. labelText => the new text, you want to set

Example Usages

```
brixxApi.setLabelText("myCheckBox", "Hello World");
```

setTooltipText

Sets the tooltip text for a control at runtime.

Parameters

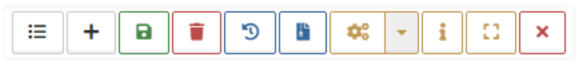
1. controlId => Id of the control
2. text => the new text, you want to set

Example Usages

```
brixxApi.setTooltipText("myCheckBox", "Hello World");
```

setToolbarButtonMode

A toolbar is a set of tools. These tools are assigned to each app present in a workspace in Brixxbox. A general toolbar for an app looks like this.



Starting from left to right. The first button shows the list of records of current app. Second one (plus sign) open a new window of current app to add a new record. Third one is a save button, after adding all details in the app, user needs to click save button to permanently store the record. If the user presses "plus" button after adding required details, the record is not save but a new window will be opened so user should always press save button. Fourth button is a delete buton, as clear from name, it is used for deleting already present records. Next button is "History record" button, it records all the history of records being added.

The sixth button is "File Attachments" button. It is not present in the toolbar by default, system user has to enable it by going into app properties, scrolling down and clicking on "Attachments" checkbox. Next button is "settings" button, on clicking Brixxbox will open current app in edit or development mode with controls on left side and live app panel on right side. Next button shows the documentation available related to current app. Second last button is for openeing current app in a stand alone mode, it is a mode where only this app covers whole screen. Last button (x) is for closing the app. There are relative symbols present on each button so it is easier for user to be able to understand what each button does.

BrixxApi in its **setToolbarButtonMode** function provides the functionality to **enable, disable or hidden**. This is helpful in restricting access, functionality to different users, for example end user should not be allowed to delete any record so delete tool bar button should be hidden from these users.

Example

```
brixxApi.setToolbarButtonMode("brixxToolbarNewRecord", "hidden");
```

Parameters

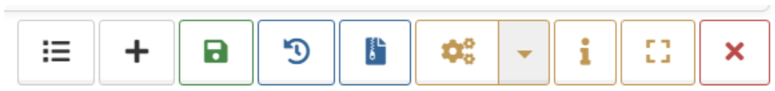
1. toolbarButtonId - This is first parameter. It is build in button ids assigned by Brixxbox to each toolbar button. These id are as follows:
    - brixxToolbarList - For listing records button.
    - brixxToolbarNewRecord - For adding records button.
    - brixxToolbarSaveRecord - For saving records button.
    - brixxToolbarDeleteRecord - For deleting records button.
    - brixxToolbarHistory - For checking records history button.
    - brixxToolbarDiscussion - For discussion button.
    - brixxToolbarAttachments - For adding attachments button.
    - brixxToolbarEditConfigNewTab - For opening app in edit panel button.
    - brixxToolbarShowAppWiki - For show wiki documents button.
    - brixxToolbarOpenFullSize - For opening app in full size button.
2. mode - This is second parameter. It sets the state of respective button. These states are as follows:
    - "disabled" (or false or not clickable)
    - "enabled" (or true or clickable)
    - "hidden"

Example Usages

System user needs to pass button id and mode as a string parameters to this function. Lets suppose system user wants to hide a delete button from end users, there are many ways to do it. Let do it on an event when this app starts. Now we need to go to app properties and add event "onAppStart". Here we will place our code to hide delete button.

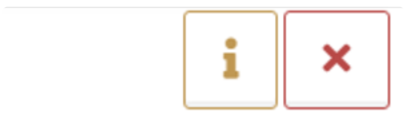
```
brixxApi.setToolbarButtonMode("brixxToolbarDeleteRecord", "hidden");
```

After this save the app and try it. Now to toolbar should look like this:



Lets now hide all the buttons available in the toolbar on app start. For this we need to add below code to an event "onAppStart" custom code.

```
brixxApi.setToolbarButtonMode("brixxToolbarList", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarNewRecord", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarSaveRecord", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarDeleteRecord", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarHistory", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarDiscussion", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarAttachments", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarEditConfig", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarEditConfigNewTab", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarShowAppWiki", "hidden");
brixxApi.setToolbarButtonMode("brixxToolbarOpenFullSize", "hidden");
```



As you can see all the toolbar buttons are hidden except the "info" button. This is the default functionality.

setEnable

Enables or disables a specific control.

Parameters

1. controlId - id of the brixxbox control as a string
2. enable - (optional, default = true) bool value if the control should be enabled or not
3. subControl (optional) Some controls have sub Controls, that can be enabled or disabled. **Important: setEnable for subcontrols cannot be used in onAppStart, because some controls are not fully created at that time, use onAppInitialized instead**. Here is a list of possible subControls:
   - grids
     - inline - enables or disables the inline editing of a grid (Cell Editing)
     - buttons - enables or disables the buttons of a grid
     - buttonNew - enables or disables the new buttons of a grid
     - buttonDelete - enables or disables the delete buttons of a grid
     - buttonCopy - enables or disables the copy buttons of a grid

Example Usages

1. Simple

```
brixxApi.setEnable("myControlId"); //enables the control
brixxApi.setEnable("myControlId", true); //enables the control
brixxApi.setEnable("myControlId", false); //disables the control
```

2. Disable the Grid inline edit feature

```
brixxApi.setEnable("myGridControl", false, "inline");
```

3. Disable all data manipulating grid Toolbar Buttons

```
brixxApi.setEnable("myGridControl", false, "buttons");
```

4. Disable a single grid Toolbar Buttons

```
brixxApi.setEnable("myGridControl", false, "buttonNew");
brixxApi.setEnable("myGridControl", false, "buttonDelete");
brixxApi.setEnable("myGridControl", false, "buttonCopy");
```

setVisibility

Sets the visibility for a specific control.

Parameters

1. controlId - id of the brixxbox control as a string
2. visible - (optional, default = true) bool value if the control should be visible or not
3. subControl (optional) Some controls have sub Controls, that can be set visible or hidden. **Important: setVisibility for subcontrols cannot be used in onAppStart, because some controls are not fully created at that time, use onAppInitialized instead**. Here is a list of possible subControls:
   - grids - The grid toolbar has to be active to see buttons at all!
     - buttons - shows or hides the buttons of a grid
     - buttonNew - shows or hides the new buttons of a grid
     - buttonDelete - shows or hides the delete buttons of a grid
     - buttonCopy - shows or hides the copy buttons of a grid

Example Usages

1. To show a control

```
brixxApi.setVisibility("myControlId");
brixxApi.setVisibility("myControlId", "visible");
brixxApi.setVisibility("myControlId", true);
```

2. To hide a control

```
brixxApi.setVisibility("myFieldId", false);
```

3. Hide the grid new button

```
brixxApi.setVisibility("myGrid", false, "buttonNew");
```

startBrixxbox

Starts a brixxbox app.

Parameters

1. startOptions - This is a json object with the start parameters:
   - appName - name of the config as string
   - noInitialRefresh - the grids of the target apps are not refreshed on startup. Can be used if you send additionalValues for the selection and you want to avoid to fetch data, that will be imidiately replaced with the correct filtered data.
   - parentApp - the parent for the new api. If not set, this will be set to brixxApi from the caller, your current api.
   - id - (optional) id of the record to load in the new app
   - startMode - (optional)
     - "modal" will start in a modal overlay window (default),
     - "tab" will start the app in a new tab. You will not get a brixxboxApi object as a result in this case!
     - "page" will replace the content in the current browser tab
     - "invisible" will not show the app
     - "popup" will start as modal in a movable and resizable window
   - appMode - (optional)
     - "form" will start the app as a Form (default),
     - "list" will start the app as a List of Records,
   - additionalValues (optional): json object of control values that should be set in the new app
   - parameters: (optional): json array of parameter flags (the flags you can choose in the menu endpoint editor) to set for the new app
   - container: (optional): a jQuery object to a div, where the new brixxbox will be embedded

Return value

If a brixxbox is started in "modal" mode, the function returns the brixApi object of the new app. You have to **await** the result (Example 3) to use it.

Example Usages

This will start the app "myapp" in a modal dialog.

```
brixxApi.startBrixxbox({appName: "myapp"}); //starts myapp as a form

brixxApi.startBrixxbox({appName: "myapp", appMode:"list"}); //starts myapp as a list
```

This will start the app "myapp" in a modal dialog, load the record with id 1 and set the values of the 2 controls "addressNo" and "orderNo" inside the new app

```
let startOptions = {
   appName: "myapp",
   id: 1,
   startMode: "modal",
   additionalValues: {
      addressNo: 123,
      orderNo: 456
   },
   parameters:["openOrders", "shippedORders"]

};
brixxApi.startBrixxbox(startOptions);
```

This will start the app "myapp" in a modal dialog, and add an addEventListener for the save event.

```
let modalBrixxApi = await brixxApi.startBrixxbox({appName: "myapp"});

modalBrixxApi.addEventListener("onRecordSaved", function (modalApi, eventArgs) {
   alert("Record saved in " + modalApi.appName);
});
```

startScanner

Brixxapi allows users to scan barcodes or QR codes with **startScanner** function. It takes two parameters: control id, it is used to identify on which control the code value will be assigned, and options object in which we pass the bool value for continuous mode. Continuous mode, if true, provides the ability to keep the scanner open after reading code otherwise it will close. The process of reading codes consists of two steps. First step is to use this function. This function needs to be placed under a "Scanner" or "Wedge scanner" controls. When an event is triggered, this function will be called and it will access the camera of the device(if available). Now when it will read a code, this value will be assigned to scanner control in an app. In the second step, we can use this value by using **getFieldValue** function . It will be used to identify the object depending upon the code and a valueU Starts scanning QR or barcodes with the given scannerControl.

```
app.startScanner("myScannerControl");
```

Parameters

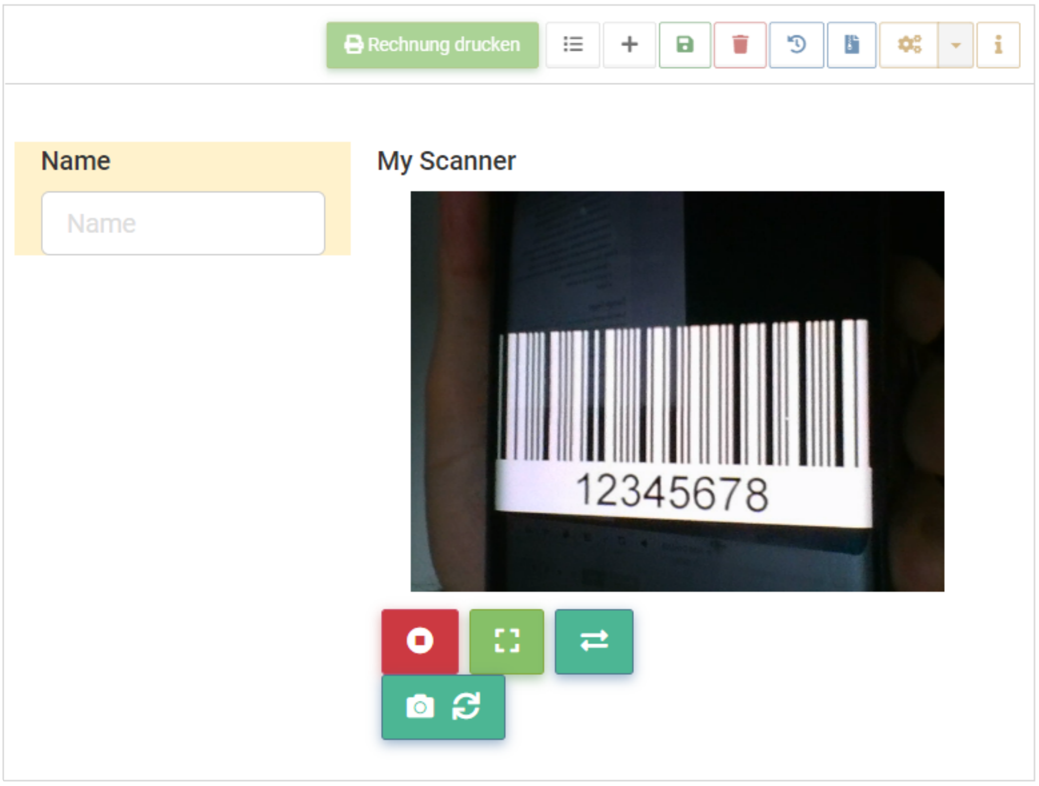
1. controlId(String) - The control id of the scanner control (Check scanner control properties in an app).
2. scanOptions(JSON Object) - This is a json object with the scanner parameters(Used to specify the scanner functionality):
   - continuousMode(Bool - Optional - Default: false) - If true, it will keep the scanner open, after reading a code.

Example Usages

Start scanner with JSON options

```
app.startScanner("myScannerControl", {
    continuousMode: true
});
```

Here we are scanning a simpled bar code with a value of '12345678'.

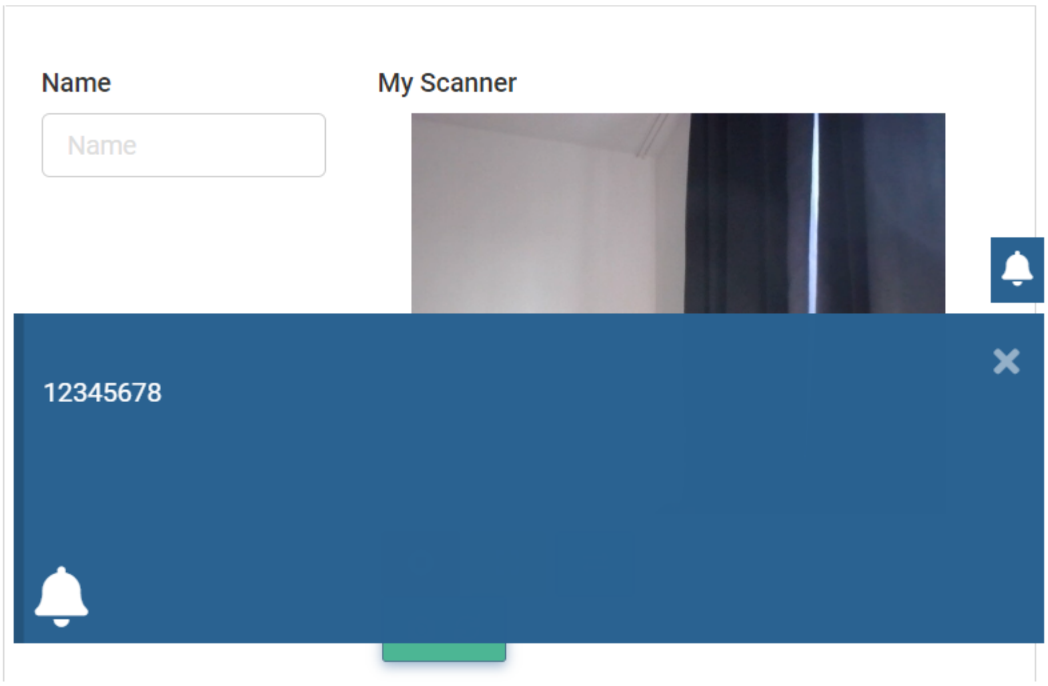


After scanning the code successfully, the value will be assigned to the scanner control present in current app. Now there are many ways to use this value e.g conditionals (if statements). For this tutorial, we will just show the scanned value as a notification by using the code snippet given below. As soon as the scanner scans the code it triggers an "onScan" event. In this event, we can add our customized code.

Code when onScan Event get triggered

```
app.showMessage({
    content: app.getFieldValue("myScannerControl")
});
```

As we can see, it is showing the scanned value of '12345678'.



12345678

showBlob

Shows a blob document in a docViewer control.

Parameters

1. controlId - The id of a "docViewer" control
2. blobData - The blob data.

Example Usages

```
    let myBlob = brixxApi.getAttachmentByFileName("Invoice1.pdf");
    brixxApi.showBlob("myDocViewer", myBlob);
```

showMessage

Displays a message to the user.

Parameters

1. messageOptions- This is a json object with the message parameters:
   - content - this is the message Text
   - title - (optional) id of the record to load in the new app
   - timeout - (optional) a timeout in milliseconds. The message will disappear after this timeout. No timeout by default.
   - "mode" - (optional) will start in a modal overlay window (default),
     - "big" - (default) a small box in the lowerright corner of the page
     - "small" - a small box in the upper right corner of the page
   - icon - (optional) a icon class string (default: "fa fa-bell swing animated")
   - sound - (optional) play a sound (true) or no (false = default)
   - type - (optional) PReset color scheme. You can use all the button types (danger, success, warning...)
   - buttons - (optional, only for mode "big") You can add buttons to the message. Just provide a list with each button in square brackets (e.g. "[Yes][No][Cancel]"). For more details please check the examples.
   - callback - (optional, only for mode "big") If u provide more than one button the callback option is needed to distinguish which action needs to be taken for a button. For more details please check the examples.

Example Usages

Most simple message

```
    brixxApi.showMessage({content: "Message Text" });
```

using additional options

```
    brixxApi.showMessage({
             title: "My Title",
             content: "My Text",
             type: "danger",
             sound: true,
             timeout: 2000,
             icon: "fa fa-bell swing animated"
            });
```

message with buttons

```
//If u provide more then one button the callback option is needed to distinguish wich action needs to be taken for a button.

function msgCallBack (reasonForCallBack){
   // possible values for reason:
   // "click" : the message was closed with the X-button
   // "timeout" : the timeout elapsed
   // "btn1", "btn2", ...  : each button gets an id number (starting from 1). e.g. if the second button was pressed -> "btn2"

   console.log("reasonForCallBack:" + reasonForCallBack);
}

brixxApi.showMessage({
         title: "My Title",
         content: "My Text",
         type: "info",
         sound: true,
         icon: "fa fa-bell swing animated",
                     buttons: "[Yes][No][Cancel]",
                     callback: msgCallBack,
        });
```

message with buttons, timeout and changing the layout of the message after it was created

```
function msgCallBack (reasonForCallBack){
   console.log("reasonForCallBack:" + reasonForCallBack);
}

let msgId = brixxApi.showMessage({
         title: "My Title",
         content: "My Text",
         type: "info",
         sound: true,
         timeout: 20000,
         icon: "fa fa-bell swing animated",
         //when using the timeout option the running timer value can be shown on one button. This
         //will visualize the default action taken after the timeout elapsed. Just add "{timer}" to a button text.
         buttons: "[ <i class='fas fa-check'></i>  Yes {timer}][ <i class='fas fa-do-not-enter'></i>  No]",
         callback: msgCallBack,
        });

 //each message will get an unique id number. This can be used to add/change the message layout further.
 document.getElementById('bigBoxBtn1-Msg' + msgId).classList.add("btn-success");
 document.getElementById('bigBoxBtn2-Msg' + msgId).classList.add("btn-danger");
```

showAttachments

opens the attachment sidebar with the attachment for the current record.

Parameters

show - open(true = default) or closes(false) the attachment sidebar. You don't have to worry if it is opend or closed before, but show will also refresh an open attachment.

Example Usages

```
brixxApi.showAttachments(); //open the sidebar
```

```
brixxApi.showAttachments(true); //open the sidebar
```

```
brixxApi.showAttachments(false); //closes the sidebar
```

showDiscussion

Opens the discussion sidebar with the chat/discussion for the current record.

Parameters

show - open(true = default) or closes(false) the discussion sidebar. You don't have to worry if it is opend or closed before, but show will also refresh an open discussion.

Example Usages

```
brixxApi.showDiscussion(); //open the sidebar
```

```
brixxApi.showDiscussion(true); //open the sidebar
```

```
brixxApi.showDiscussion(false); //closes the sidebar
```

showWikiPage

Displays a wiki page in the right sidebar.

Parameters

1. pageName - The name of the wikie page (e.g. "function_showWikiPage")
2. global - Set this to true, if you want to display a page from the global wiki instead of a wiki page from your workspace wiki. The default is global=false.

Example Usages

Global documents

```
brixxApi.showWikiPage("function_setVisibility", true);
```

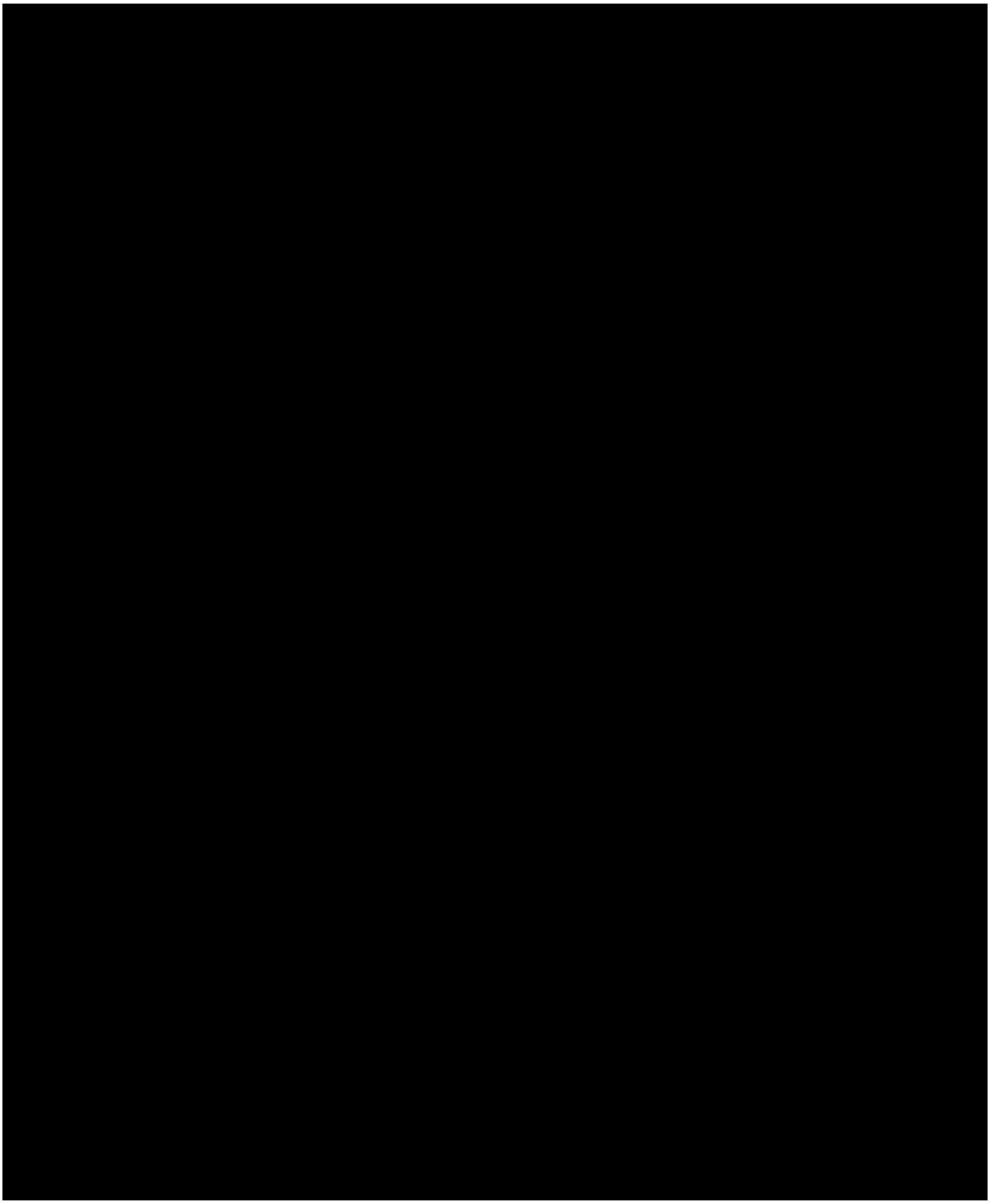
Workspace documents

```
brixxApi.showWikiPage("appConfigManual_address", false);
brixxApi.showWikiPage("appConfigManual_address"); //2. parameter is false by default, so both lines give the same result
```

sqlReadValue

Reads a single value from a standard sql statement. See SQL Statements for how to create statements.

Demo



Parameters

1. statementName - The name of the standard statement
2. additionalParameters - (optional) all controls in the current config are set as paramters automatically. If you need to add additional parameters, yo
3. columnName - (optional) the column name to pick from the first result line. If this is empty, the first column will be used
4. queryOptions - (optional) a json object with options for the request
   - timeout - (optional) timeout for the SQL Request. Default is 30 seconds
   - connectionKey - (optional) a key to a custom settings entry with the connection string to an external MSSQL database

Example Usages

Lets assume, we have a standard statement "select adrName, adrStreet from address where id=@id"

```
    let result = await brixxApi.sqlReadValue("readAddress"); //this will use the statement "readAddress". The @id parameter is set
    console.log(result);
```

```
    let result = await brixxApi.sqlRead("readAddress", {id: 1}); // as above, but we overwrite the id from the app with a specific
    console.log(result);
```

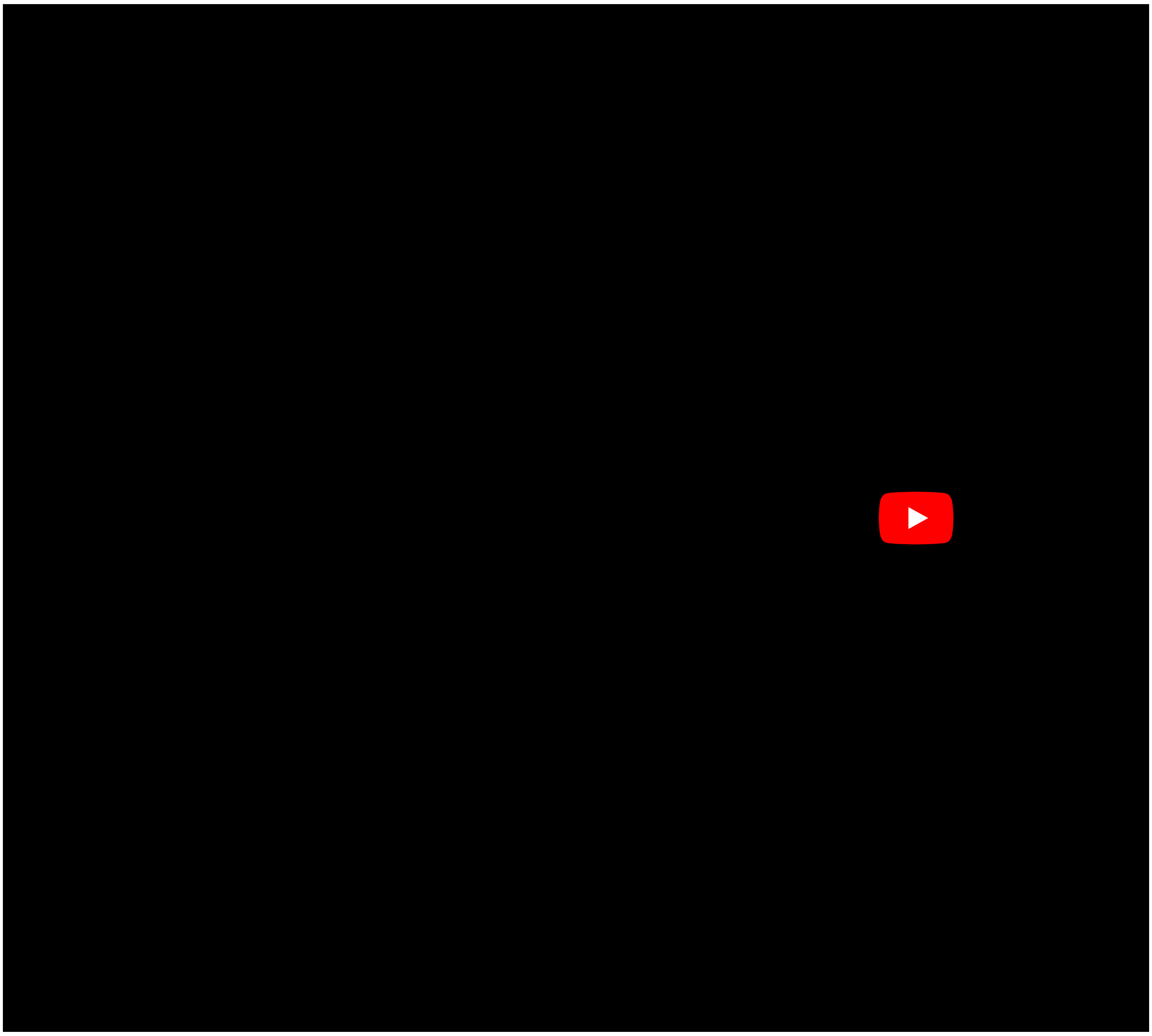
```
    let result = await brixxApi.sqlRead("readAddress", {id: 1}, "adrStreet"); // as above, but we overwrite the id from the app wi
    console.log(result);
```

```
    let result = await brixxApi.sqlRead("readAddress", {id: 1}, "adrStreet", {timeout: 45}); // as above, but with 45 seconds time
    console.log(result);
```

sqlWrite

Uses a standard update or insert statement. See SQL Statements for how to create statements.

Demo



Parameters

1. statementName - The name of the statement
2. additionalParameters - all controls in the current config are set as paramters automatically. If you need to add additional parameters, you can use
3. queryOptions - (optional) a json object with options for the request
    - timeout - (optional) timeout for the SQL Request. Default is 30 seconds
    - connectionKey - (optional) a key to a custom settings entry with the connection string to an external MSSQL database

Example Usages

Lets assume, we have a standard statement "update address set adrName = @newName"

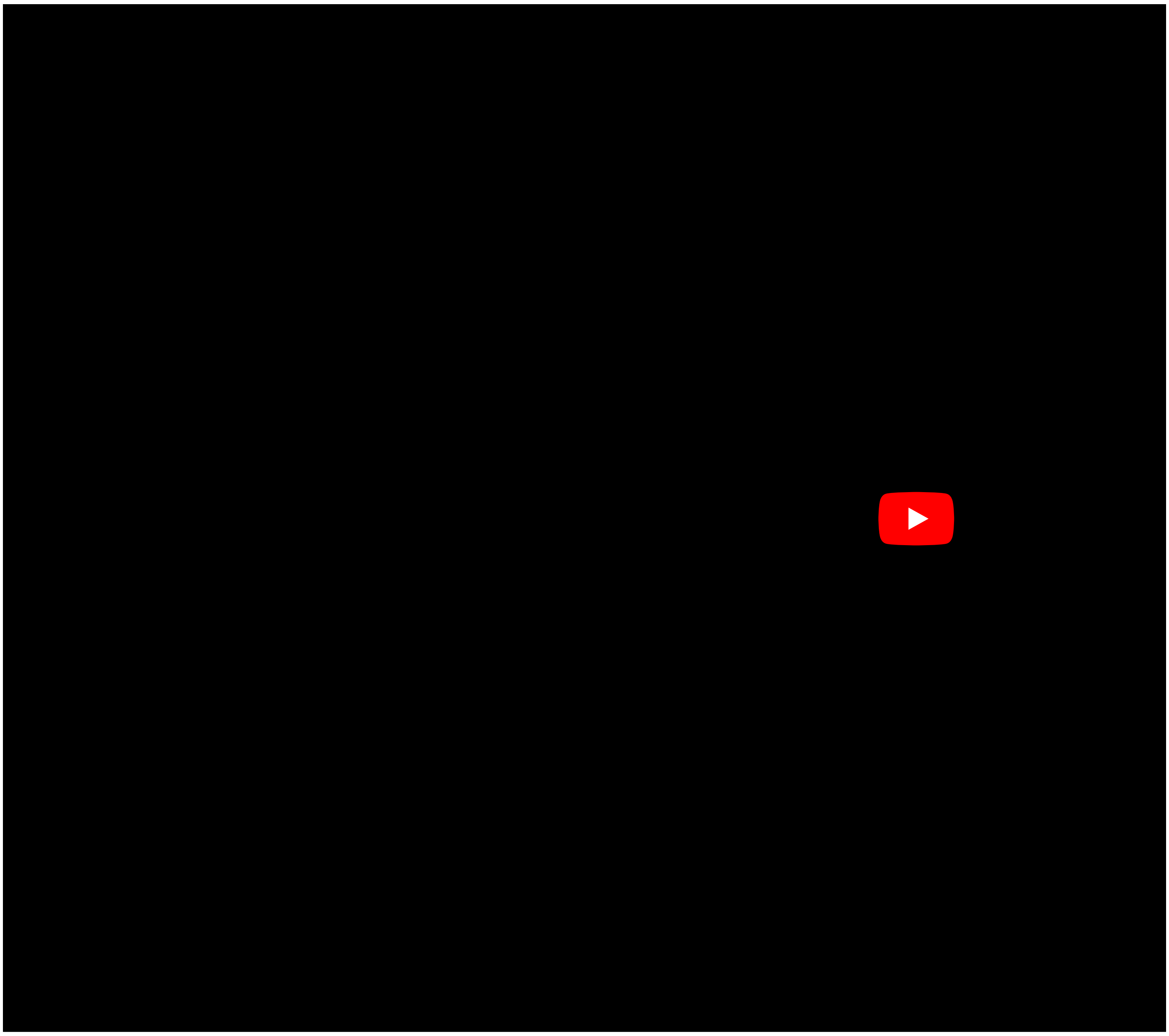
```
let result = await brixxApi.sqlWrite("updateAddress"); //this will use the statement "updateAddress". The @newName parameter
console.log("Rows changed: " + result);
<syntaxhighlight>

<syntaxhighlight lang="js">
let result = await brixxApi.sqlWrite("readAddress", {newName: "Hello World"}); //this will use the statement "updateAddress".
console.log("Rows changed: " + result);
```

```
let result = await brixxApi.sqlWrite("readAddress", {newName: "Hello World"}, {timeout: 45}); //this will wait for 45 seconds
console.log("Rows changed: " + result);
```

sqlRead

Reads data from a Standard Sql Statement. See SQL Statements for how to create statements.

Demo



Parameters

1. statementName - The name of the statement
2. additionalParameters - all controls in the current config are set as paramters automatically. If you need to add additional parameters, you can use
3. queryOptions - (optional) a json object with options for the request
   - timeout - (optional) timeout for the SQL request. Default is 30 seconds
   - connectionKey - (optional) a key to a custom settings entry with the connection string to an external MSSQL database

Example Usages

Lets assume, we have a standard statement "select adrName from address where id=@id"

```
    let result = await brixxApi.sqlRead("readAddress"); //this will use the statement "readAddress". The @id parameter is set to t
    console.log(result[0].adrName);
```

```
    let result = await brixxApi.sqlRead("readAddress", {id: 1}); //this will use the statement "readAddress". The @id parameter is
    console.log(result[0].adrName);
```

```
    let result = await brixxApi.sqlRead("readAddress", {id: 1}, {timeout: 60}); // Wait for 60 seconds
    console.log(result[0].adrName);
```

```
    let result = await brixxApi.sqlRead("readAddress", null, {timeout: 60}); //use a timeout but without script parameters
    console.log(result[0].adrName);
```

saveCurrentRecordWithoutEvents

Works exactly like saveCurrentRecord. Except, that it des not trigger any events (like close on save).

Example Usages

Simple (pay attention to the **await** keyword)

```
    await brixxApi.saveCurrentRecordWithoutEvents();
```

saveCurrentRecord

Saves the changes to the current record in the database.

Example Usages

Simple (pay attention to the **await** keyword)

```
await brixxApi.saveCurrentRecord();
```

saveConfigRecord

Saves a record for a config

1. configName - name of the app config
2. record - a record structure Json object (as it comes from loadConfigRecordById)

Example Usages

Simple:

```
let record = await brixxApi.loadConfigRecordById("myConfig", 1);
record.data.Name = "John Doe";
brixxApi.saveConfigRecord("myConfig", record);
```

Create a new record:

```
brixxApi.saveConfigRecord("myConfig", {
    data: {
        myFirstName: "John",
        myLastName: "Doe"
    }
});
```

Create a new record and use the result:

```
let newRecord = brixxApi.saveConfigRecord("myConfig", {
    data: {
        myFirstName: "John",
        myLastName: "Doe"
    }
});
console.log("New Record Id:" + newRecord.data.id);
```

setBackgroundColor

Sets the background color of a control.

Parameters

1. controlId
2. colorName - use one of these: "default", "primary", "secondary", "success", "danger", "warning"

Example Usages

Set to warning

```
brixxApi.setBackgroundColor("myControl", "warning");
```

Set to normal

```
brixxApi.setBackgroundColor("myControl", "default");
brixxApi.setBackgroundColor("myControl"); //"default" is "default" ;)
```

setFontStyle

Sets the font style of a control.

Parameters

1. controlId
2. fontStyle - use one of these: "italic", "normal"

Example Usages

Set to italic

```
brixxApi.setFontStyle("myControl", "italic");
```

Set to normal

```
brixxApi.setFontStyle("myControl", "normal");
brixxApi.setFontStyle("myControl"); // you can skip param2 if you want to set back to normal
```

setFontWeight

Sets the font weight of a control.

Parameters

1. ControlId
2. FontWeight - one of those: "normal", "bold"

Example Usages

Set to bold

```
brixxApi.setFontWeight("myControl", "bold");
```

Set to normal

```
brixxApi.setFontWeight("myControl", "normal");
brixxApi.setFontWeight("myControl"); //you can skip param2 if you want to set it back to normal
```

setGridDefaults

brixxApi.SetGridDefaults

Sets the grid to its default values. Any setting for this grid for the user will be lost.

Parameters

1. controlId - id of the grid control

Example Usages

```
brixxApi.SetGridDefaults(myGrid);
```

setTextColor

Sets the text color for a control to one of the predefined colors

Parameters

1. controlId - the id of the control to change the color
2. colorName - one of these colors: "default", "primary", "secondary", "success", "danger", "warning"

Example Usages

Set Color to "warning"

```
    brixxApi.setTextColor("myControl", "warning");
```

Set Color to default

```
    brixxApi.setTextColor("myControl");
```

setFocus

Sets the cursor focus to a specific control and optionally selects the existing text.

Parameters

1. controlId - the name of the control to set the focus to
2. select - (optional) if set to true, the existing text in the control is selected. if empty or false, the cursor is placed behind the not selected text

Example Usages

```
    brixxApi.setFocus("adrName"); //set focus and palce cursor after existing text
    brixxApi.setFocus("adrName", true); //set focus and select the existing text
```

setDecimalDigits

Change the decimal digits of a numeric field.

Parameters

1. controlId - Id of the control
2. The value of digits for the numeric field.

Example Usages

Set digits

```
brixxApi.setDecimalDigits("myControlId", 3);
```

switchTagControl

Changes the tag control to edit mode and vice versa.

Parameters

1. controlId - String with the control id.
2. editMode - Optional boolean to determine if the control should be in edit mode or not. If not stated, the control will switch to it's other mode.

Example Usages

```
brixxApi.switchTagControl('fieldId');
```

```
brixxApi.switchTagControl('fieldId', true);
```

selectGridRows

Selects all rows with a specific value in a data grid column.

Parameters

1. controlId - control id of the grid
2. columnId - id of the column to compare the value with
3. value - if the row has this value in the column with "columnId", it will be selected

Example Usages

```
brixxApi.selectGridRows("myGrid", "imeiNumber", 1234567);
```

showRowDetailPanel

This function is used to toggle the row detail panel of a grid row.

Parameters

1. Row - the row object. Usually this is: eventArgs.details.row
2. show - shows (true, default) or hides (false) the detail Panel

Example Usages

This code belongs in an **onRowCreated** event

```
    app.showRowDetailPanel(eventArgs.details.row, eventArgs.details.data.icordlnIsBillOfMaterial===true);
```

showRowDetailButton

This function is used to toggle the row detail button, that opens the child app, based on the row data.

Parameters

1. Row - the row object. Usually this is: eventArgs.details.row
2. show - shows (true, default) or hides (false) the detil Button

Example Usages

This code belongs in an **onRowCreated** event

```
app.showRowDetailButton(eventArgs.details.row, eventArgs.details.data.icordlnIsBillOfMaterial===true);
```

setGridAutoRefresh

Sets the auto refresh interval for grids. Only valid for grids with this setting enabled.

Parameters

1. controlId - id of the grid control
2. autoRefreshSeconds - The value can be 0 to switch auto refresh off. Any other value must align with the defined values for this grid. For settings

Example Usages

```
    brixxApi.setGridAutoRefresh(myGrid, 0); //this will switch the auto refresh off
    brixxApi.setGridAutoRefresh(myGrid, 120); //this will switch the auto refresh to 120 seconds (2 minutes). The parameter is onl
```

setGridGrouping

Sets the grouping options for a grid control.

Parameters

1. controlId - the Id of the grid control
2. groupConfiguration - either null, or undefined to disable grouping, or a column (controlId)

Example Usages

```
    brixxApi.setGridGrouping(myGridControl); //Disables grouping
    brixxApi.setGridGrouping(myGridControl, productId); //groups by productId
```

showMessageBox (deprecated)

brixxApi.showMessageBox

Displays a modal messagebox with buttons. Each button gets a function that is called when the button is clicked. The Messagebox will **NOT** block the script. The next line after the messagebox will be executed while the messagebox is still visible. If you want to wait for the button click. use waitForMsgBox instead.

Parameters

messageBoxOptions - A JSON object with messageBox parameters

Example Usages

```
brixxApi.showMessageBox({
   title: "sampel title",
   text: "sampel text",
   buttons: [
      {
         title: "myButton",
         click: function(){
             console.log("myButton clicked!!!!");
         }
      }
]});
```

showTabPage

Selects a Tab Page.

Parameters

controlId - id of the tabPage that should shown

Example Usages

```
    brixxApi.showTabPage("detalsTab");
```

setFieldUnit

Add or set a unit in front of a numeric or text field.

Parameters

1. controlId - Id of the control
2. The string value of the unit you want to set. If this parameter is not set, the unit will be removed.

Example Usages

Simple textbox

```
    brixxApi.setFieldUnit("myControlId", "€");
```

Remove unit

```
    brixxApi.setFieldUnit("myControlId");
```

takePicture

This triggers the camera to take a picture

```
brixxApi.takePicture("myCameraControl")
```

triggerEvent

Triggers an events for a control, or the brixxbox app itself. Some events are on brixxbox App level (like e.g. onRecordSaved). Other events are on control level (like e.g. onClick)

Example Usages

```
    brixxApi.triggerEvent("click", "myButton"); //This triggers a button Click for the control "myButton"
    brixxApi.triggerEvent("recordSaved"); //This triggers the event, that would otherwise occur if a record was saved
```

You can use the whole event name with the "on" prefix as well

```
    brixxApi.triggerEvent("onClick", "myButton"); //This triggers a button Click for the control "myButton"
    brixxApi.triggerEvent("onRecordSaved"); //This triggers the event, that would otherwise occur if a record was saved
```

toggleGridSelection

Toggles the grid selection for a whole grid.

Parameters

1. controlId - control id of the grid

Example Usages

```
brixxApi.toggleGridSelection("myGrid");
```

unselectGridRows

Deselects all rows with a specific value in a column.

Parameters

1. controlId - control id of the grid
2. columnId - id of the column to compare the value with
3. value - if the row has this value in the column with "columnId", it will be unselected

Example Usages

```
brixxApi.unselectGridRows("myGrid", "id", 123);
```

uploadAttachment

Uploads an attachement

Parameters

- data - The data as blob
- documentTypeId (optional) - The ID of the document type to be saved
- fileName - (optional) name of the attachement

Example Usages

```
brixxApi.uploadAttachement(blobData, 1, "attachement.jpg");
```

updateTitle

Updates the title of the browser tab. You have to set brixxApi.appTitle property first and then update the title.

Example Usages

```
brixxApi.appTitle = "Hello World";
brixxApi.updateTilte();
```

validateInput

Checks the field Validation for all controls.

Example Usages

```
    brixxApi.validateInput();
```

queryStoredProcedure

Executes a stored procedure and returns the query result.

Parameters

1. procedureName - The name of the stored procedure
2. procedure parameters - If you need to add parameters, you can use this json object to set them
3. queryOptions - (optional) a json object with options for the request
   - timeout - (optional) timeout for the request. Default is 30 seconds
   - connectionKey - (optional) a key to a custom settings entry with the connection string to an external MSSQL database

Example Usages

1. Simple (pay attention to the **await** keyword)

```
let myResult = await brixxApi.queryStoredProcedure("procedureName");
```

2. With parameters (pay attention to the **await** keyword)

```
let parameterJson = {
   myCustomSqlParameter: 12345
};
let myResult = await brixxApi.queryStoredProcedure("procedureName", parameterJson);
```

3. Short version (pay attention to the **await** keyword)

```
let myResult = await brixxApi.queryStoredProcedure("procedureName", {
   myCustomSqlParameter: 12345
});
```

4. Using a timeout

```
let myResult = await brixxApi.queryStoredProcedure("procedureName", {
   myCustomSqlParameter: 12345
},{
   timeout: 45
});
```

Variables

userId

Readonly variable, that returns the email address of the current user.

Example Usages

```
console.log(brixxApi.userId);
```

recordId

Represents the id of the current record, if a record is loaded, otherwise it will return null.

Example Usages

```
if(brixxApi.recordId){
   console.log("the current record id is: " + brixxApi.recordId);
}else{
   console.log("sorry, no record loaded");
}
```

record

Represents the current record, if a record is loaded, otherwise it will return null.

Example Usages

Assuming the current app has a control with the id adrName

```
if(brixxApi.record){
   console.log("the current record id is: " + brixxApi.record.id);
   console.log("hello: " + brixxApi.record.data.adrName);
}else{
   console.log("sorry, no record loaded");
}
```

isLoadingRecord

Is true if the brixxApi is currently loading and displaying a record. Can be useful in an onChange event to tell the diffenrece beween a loaded value and a value that has ben set by the user or any other event.

Example Usages

In an onChange Event

```
if(brixxApi.isLoadingRecord){
   console.log("app is loading a record");
}else{
   console.log("app is not loading a record");
}
```

Controls

Accordion

Accordion

This is kind of a "Tab control", we can use it to host tab pages. For each "TabPage", it will create a separate expandable and collapsible region. User will be able to show and hide the controls of "Tab pages" by expanding and collapsing its region. You can use TabPages inside the "Accordion" like in a "Tabcontrol". The difference between "Accordian" and "Tab Control" is the later organizes "Tab Pages" label's on its first row. All Tab pages are accessable from this row while "Accordion" stacks each sibling Tab page one on another. Also these Tab pages are expandable and collapsible regions.

Documentation

Brixxbox app configurations allows you to add different controls in your app. "Accordion" is used as a container for multiple Tab pages. By default, all Tab pages will be collapsed. It behaves similar to "Tab control" which means from all sibling Tab pages only one will be in expandable form and other will be in collapsed form. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using this control, "Accordion" control type should be selected from drop down list. Multiple Tab page's which are placed on same level of inheritance inside a "Accordion" are known as sibling Tab pages.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. For "Accordion", this property is not important.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Styling

These properties defines how control looks like in Brixxbox.

- Enabled

- It allows two options: yes, no. If "yes" is selected, the control's input will be enabled. It has no effect in "Accordion" because "Accordion" itself is not expecting any kind of input.

- Disabled in edit mode

- There are two modes available in Brixxbox for each app: edit mode, create mode. In edit mode: user is allowed to add new controls and functionality while in create mode, on the basis of app controls, user is allowed to add a new record, edit an existing record, or list all the records. In order to edit a record, user has to list all the records using app settings and then select a specific record to edit. "Disabled in edit mode" is a checkbox field. If this option is selected then the control will not be functional while editing a previously saved record but it will be functional while creating a new record. In "Accordion" case, it will have no effect because "Accordion" is not expecting any kind of input.

- Visibility

- It will specify the visibility options available in Brixxbox. There are four options available: visible, hidden, hidden in grids, and hidden in forms. For example, if hidden option is selected then all sibling tab pages inside it, will be hidden.

- Text Color

- Brixxbox allows user to specify, text colors of controls. There are seven different options available in Brixxbox.

- Tooltip

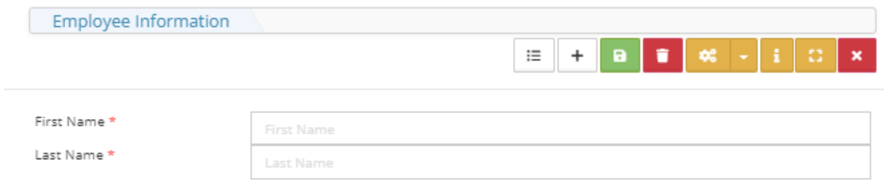
- Enter a text for a tool tip. This will be translated.

- Help Text

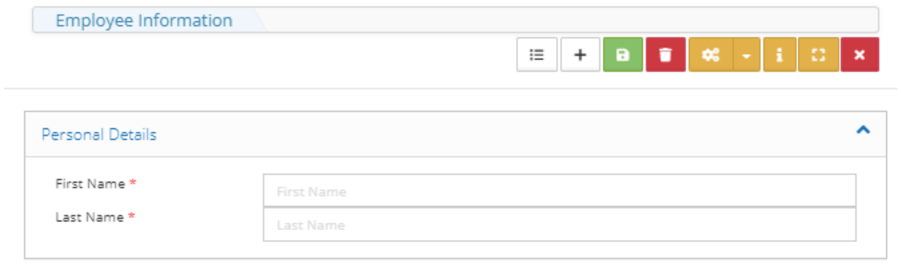
- Brixxbox with the property allow users to add some help text. This text will be displayed below the control. The purpose of this option is to encourage the right behavior of customer while using a specific control.

Tutorial

This tutorial assumes that a user has an existing app. In this app, we have two textboxes "first name", "last name". In this tutorial, we want to put them in a tab page and ultimately add this tab page in a "Accordion. Tab pages are the control containers which can be placed inside "Tab Control". If you want to know more about "Tab Page" have a look at its documentation. At start, our app looks like this:



Now first add a "Tab Page" control and label it with "Personal Details". In next step, add both textboxes "First Name", "Last Name" as child's to the tab page. Both text boxes are siblings now. In the last step, add new "Accordion" and make our tab page its child



We have successfully created a tab page and placed it inside the "Accordion". We can click on arrow (right side - top) to show and hide contents of tab page.

AppConfig

You can embed a brixxbox app into your app below a grid to show grid line details

Badge

Badge

A badge control is designed to be added as a child to button controls. It is used to show notifications over buttons. For example, total number of orders can be shown over an order button via badge.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Badge" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Refers to Config

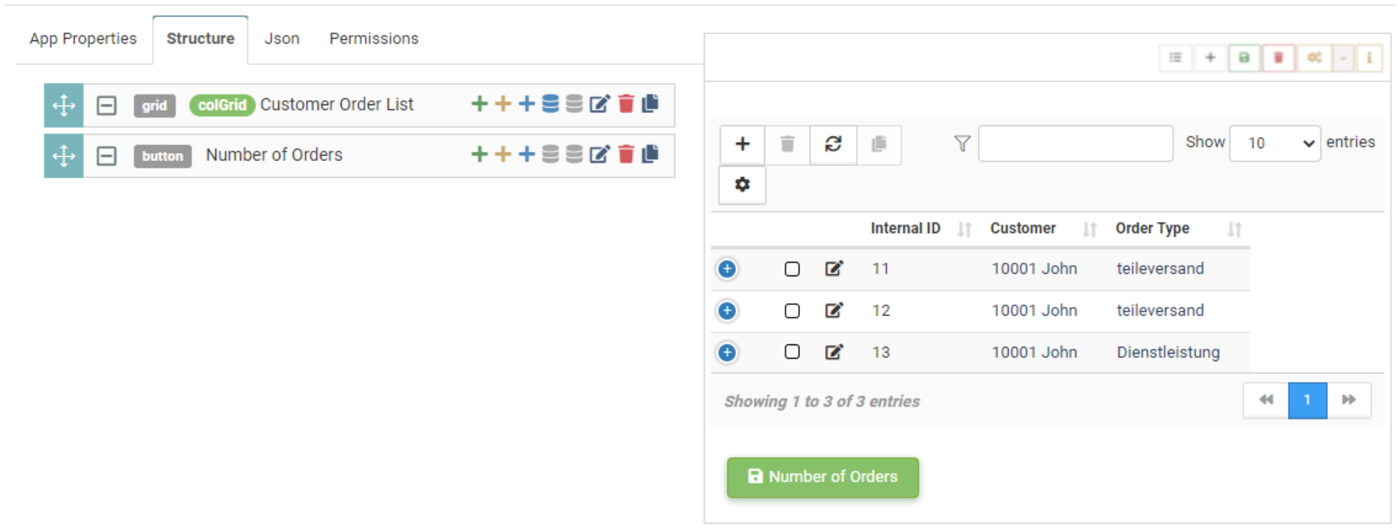
- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Default Value

- Normally this is the value which is assigned to a control when the app is opened or a new record is created, the brixxApi calls setFieldValue on this control and set the control's value with the default value. In badge, this can be the only value to get assigned to it. As it not interactable, we do not need to change its value.

Tutorial

In this tutorial, we will be using "customerOrderList" app. It is a simple app which consists of a grid and a button with a badge. The grid lists all available customer orders. The button "Number of Orders" has a badge over it which displays all the customer orders present in our system. Our app looks like this


App Properties | Structure | Json | Permissions

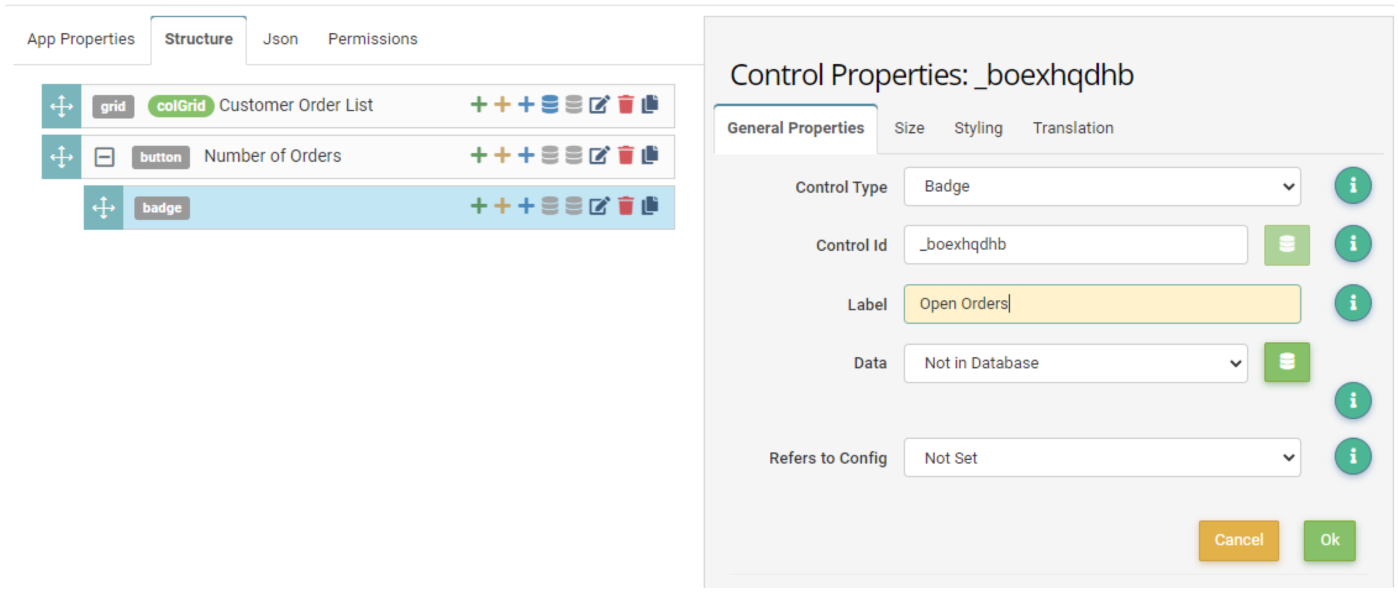
grid colGrid Customer Order List

button Number of Orders

			Internal ID	Customer	Order Type
			11	10001 John	teileversand
			12	10001 John	teileversand
			13	10001 John	Dienstleistung

Showing 1 to 3 of 3 entries

Here we can see there are only three customer orders are available. These customer orders are uniquely identified by "internal Id". For very less number of customer orders, it is easy to count them but if we have a huge amount of orders and we want their count. This count can daily, monthly, yearly or all time based. This is one scenario where we can use badge over a button. It acts like a notification over button. To add a badge we need to select a badge from top down list of controls in our app.


App Properties | Structure | Json | Permissions

grid colGrid Customer Order List

button Number of Orders

badge

Control Properties: _boexhqdhb

General Properties | Size | Styling | Translation

Control Type: Badge

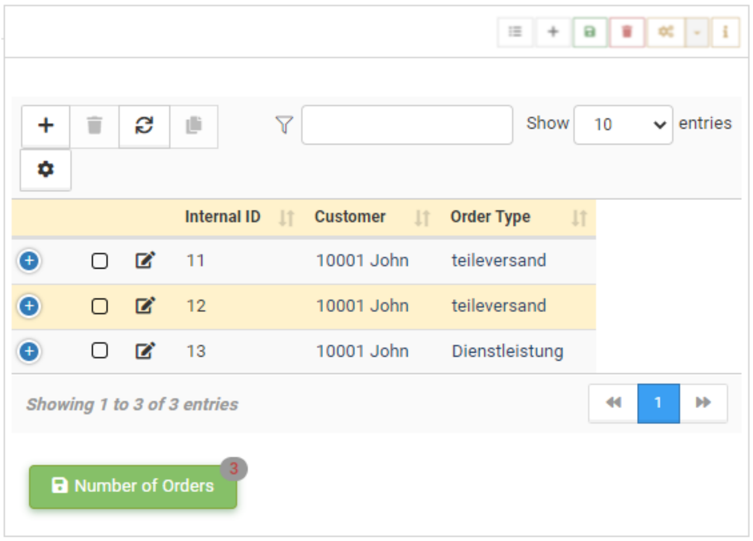
Control Id: _boexhqdhb

Label: Open Orders

Data: Not in Database

Refers to Config: Not Set

Another important point to note here is that we have placed badge as a child of the button control. You can do this by dragging the badge control towards right under button control. Our control hierarchy is shown in above figure. We care still not able to see the badge. The reason is we still need to provide data source for our badge. Lets add a data source. Click on "edit data source" button. Select "Sql" as a data source type because we want to count the total number of orders. Now we need to add a sql query which will count the number of total orders present in our system. We will use this query "SELECT count(id) from customerOrder". We are counting on the basis of "internal ids" because they are unique. Data source selection window should look like this



Show 10 entries

			Internal ID	Customer	Order Type
⊕	☐	✎	11	10001 John	teileversand
⊕	☐	✎	12	10001 John	teileversand
⊕	☐	✎	13	10001 John	Dienstleistung

Showing 1 to 3 of 3 entries

We see now a badge showing over our button and it is showing the count of total customer orders. For now we have only three customer orders in our app.

Button

Button

In Brixxbox buttons can be used to validate the business logic. For this purpose, you will most likely add a click event to specify the action.

Documentation

Brixxbox app configurations allows you to add different controls in your app. Check box is the easiest control. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using button control, "Button" control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters. For button, normally we do not want to store its value in database.

- Label

- It is the display value for control. For buttons, it makes sense to label them via action that will be performed on pressing them. For Example, on login page we see "login" or "sign up" buttons. If user is a new one then he has to sign himself up in the system by clicking on the "sign up" button and then following the default procedure. On the other hand if user is existing one, to log in he will click on "login" button.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Shortcut

- A user can assign a shortcut command to button. For example, [Ctrl]-[Alt]-D. When ever this command is executed, button will be pressed

Size

In Brixxbox, the screen is divided into 12 logical columns horizontally. For each control you can select explicitly how large it should be according to your application needs. Size properties defines how control will be displayed in Brixxbox App.

- Label on Top of Control

- If this option is selected then the label of a control and actual control will be in two different horizontal lines and control's label will be present on the top line. It does not make sense in button scenario. Button label should be displayed inside the control. It is insignificant for button but plays an important roles for other controls like textbox etc. For other controls, if this option is not selected then 12 column's will be shared between label and control itself. For example, label can be 4/12 column's long and control can be 8/12 columns long.

The display layout is divided into four different categories depending upon the pixels. For each category, Brixxbox also allows you to customize the width of control.

- Phone (and up)

- This category includes devices which have screen pixels less than 768 pixels. For examples: Smart phones. This will be by default selected category in Brixxbox. In this category, each control will be assigned to 12/12 by default. It means that in extra small screens, each control takes up the whole horizontal line and next control will take up whole next horizontal line and soon

- Tablets (and up)

- This category includes devices which have screen pixels less than 992 pixels. For example: Tablets. By switching this settings on, they will come into play for particular app.

- Medium Devices (Laptops)

- This category includes devices which have screen pixels less than 1200 pixels. For example: Laptops. In Medium devices, an input control will be set to 4/12 per control by default with option to customize the size of controls.

- Large Devices (Big Monitors)

- This category includes devices which have screen pixels greater than 1200 pixels. For example: Big monitor displays.

- Custom Label class

- This is only relevant if the control is in a From Group, and so the Label is left or right of the control instead of above it. It uses the same rules as the Width properties as above.

Styling

These properties defines how control looks like in Brixxbox.

- Enabled

- It allows two options: yes, no. If yes is selected, the control will be enabled. In button's case, it will be clickable. On the other hand, if no is selected, it will be disabled.

- Disabled in edit mode

- There are two modes available in Brixxbox for each app: edit mode, create mode. In edit mode: user is allowed to add new controls and functionality while in create mode, on the basis of app controls, user is allowed to add a new record, edit an existing record, or list all the records. In order to edit a record, user has to list all the records using app settings and then select a specific record to edit. "Disabled in edit mode" is a checkbox field. If this option is selected then the button will not be functional in edit mode but it will be functional in create mode of Brixxbox.

- Visibility

- It will specify the visibility options available in Brixxbox. There are four options available: visible, hidden, hidden in grids, and hidden in forms.

- Text Color

- Brixxbox allows user to specify, text colors of controls. There are seven different options available in Brixxbox.

- Button Style

- This option gives color to control's background This is important option for button. It gives a hint for what to expect when clicking the button. For example, a cancel button should be of red background. This gives a hint to user that in case of pressing you will lose some information.

- Squared

- This option will create a squared button. For example, if we have selected a button of 4/12 size and this option is also selected then a button of size 484 will be created.

- Icon

- Brixxbox is very dynamic and allows a lot of configurability. Many icons are present in Brixxbox and user can choose according to their needs. If user select this option then the Icon will be placed next to the button label.

- Tooltip

- Enter a text for a tool tip. This will be translated.

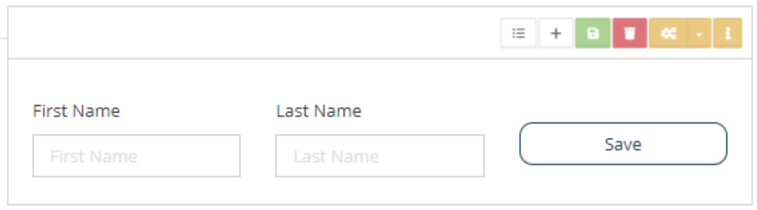
- Help Text

- Brixxbox with the property allows users to add some help text. This text will be displayed below the control (in this case button). The purpose of this option is to encourage the right behavior of customer while using a specific control.

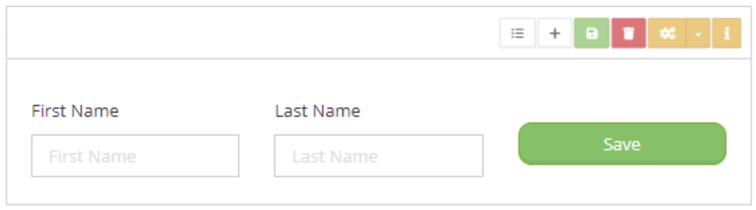
For further information please have a look at Bootstrap Grid System

Tutorial

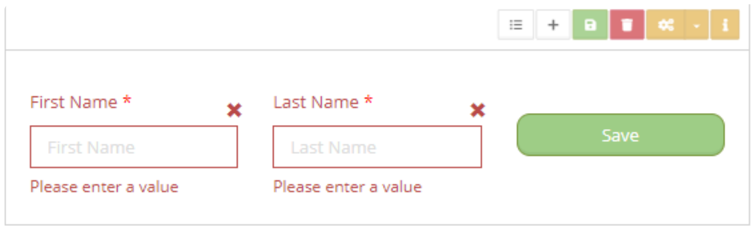
This tutorial assumes that a user has an existing app and they want to add a new control of type "Button". In this app, we have two textboxes "first name", "last name" and we want to add a "save" button. We want to validate the inputs and save them when button is pressed. First of all, select "Button" from list of controls. After that you need to give an id to your button. After this save your application. Your application should look like this:

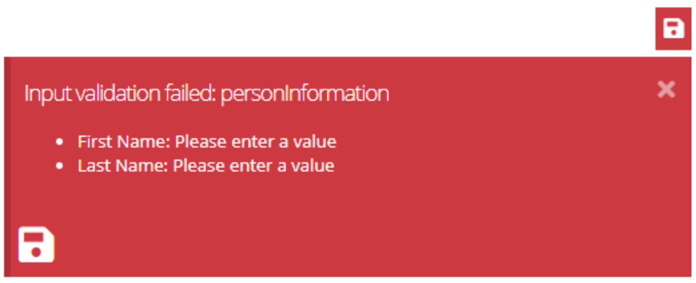


We can see a button named "Save" with white back ground. This does not give look promising. Now we can add different background colors. For doing this, go to button edit control and select "styling" tab. Here choose "Success" from Button sytle property. Now save your app. It will now look like this



We want to save the employee first name and last name when the button "Save" is pressed. Firstly, we need to add "onClick" event to our button and then we need to add code to our button "onClick" event. When our button is pressed, firstly we need to validate the input and display error if not. Add "nonEmpty" control validator on both "First Name" and "Last Name". Secondly, add "app.saveCurrentRecord();" code to "onClick" of our save button. Now if you try to save the empty record. It will give error. The error is shown below:





Input validation failed: personInformation

- First Name: Please enter a value
- Last Name: Please enter a value

Camera

This control starts the mobile camera to take a picture. For desktop browsers you will be able to upload a picture.

Special properties

- Checkbox 'Add to attachements' - This will automatically add the Picture to the attachements of the current record

brixxApi Functions

- takePicture("controlId") - Triggers the camera

Events

- onPictureTaken - Fires everytime a picture is taken/uploaded.

Example usage for onPictureTaken

Print picture

```
 brixxApi.printBlob(brixxApi.getFieldValue("controlId"));
```

Add picture to image control

```
brixxApi.setFieldValue("controlId image control", brixxApi.getFieldValue("controlId camera control"));
```

Add picture to attachements

```
brixxApi.uploadAttachement(brixxApi.getFieldValue("controlId"), documentTypeId(optional), filename(optional))
```

Chart

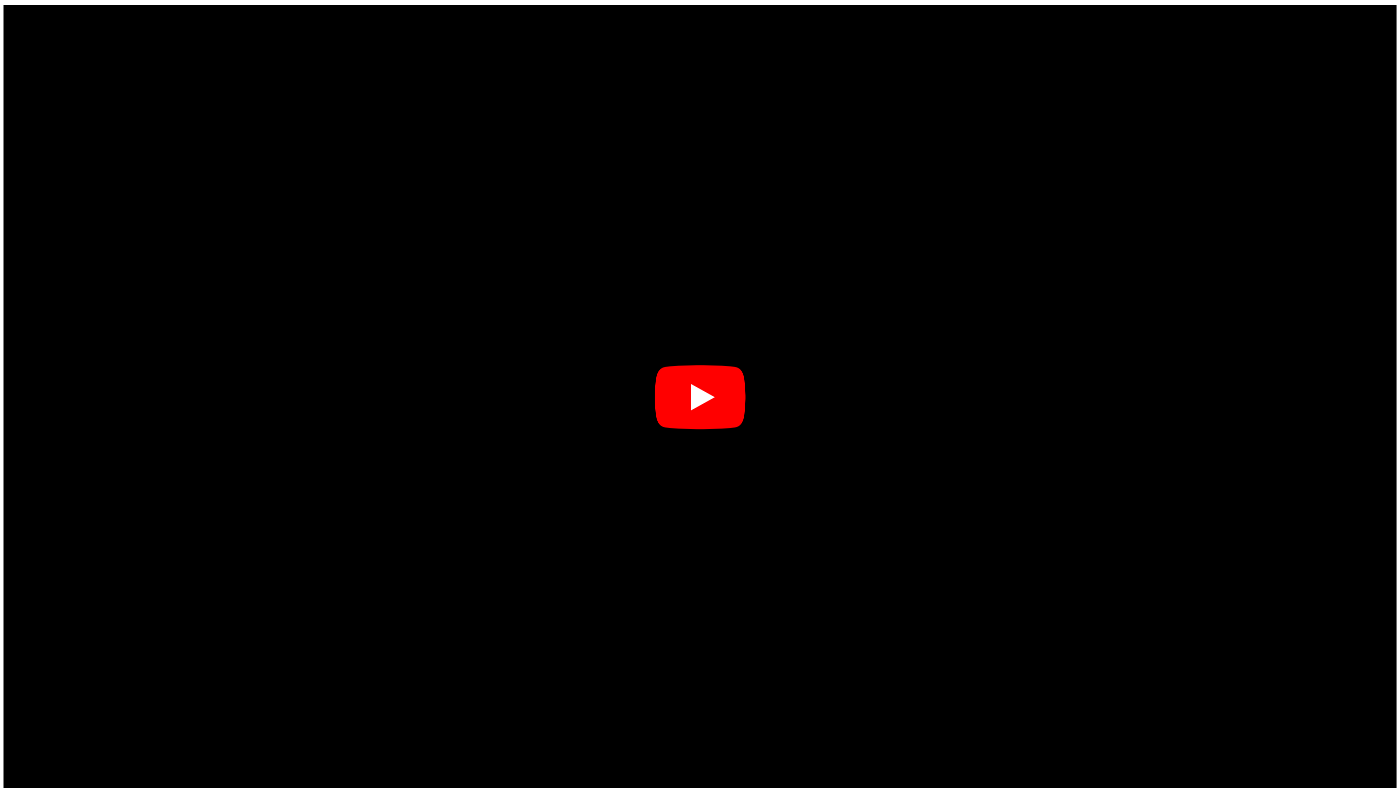
A Control to display bar, line or pie charts in your app.

CheckBox

CheckBox

A checkbox that can be checked (value: 1) or not (value: 0). For example, if you have an items application. You can use checkbox to show its availability.

Demo



Documentation

Brixxbox app configurations allows you to add different controls in your app. Check box is the easiest control. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using Check box control, "Check Box" control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters. Brixxbox saves checkbox value in one bit.

- Label

- It is the display value for control. For checkboxes, it makes sense to label them like "isAvailable" in the form of Yes/ No questions

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

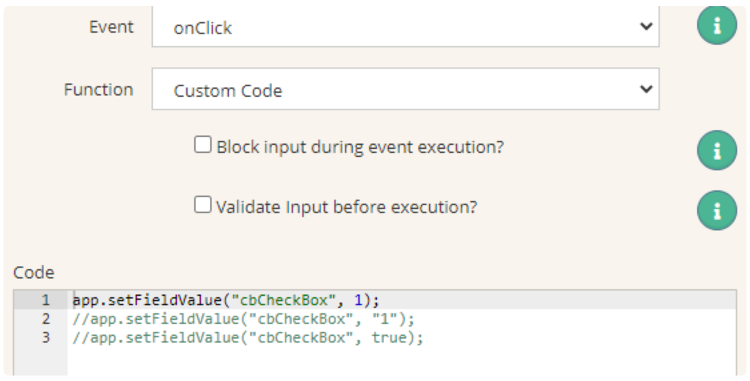
- Default value

- This value is set when an application is opened or a new record is created.

Tutorial

This tutorial assumes that a user has an existing app and they want to add a new control of type "Check Box". First of all, select "Check Box" from list of controls. After that you need to give an id to your check box and create column in database. We also added two buttons "Set" and "Reset". For this tutorial we will follow the structure of demo given above.

We need to add code to our button "onClick" events. "Set" button will set the value of checkbox and "Reset" button will unselect it. Brixxbox accepts input for setting/resetting checkboxes in following formats: as an integer (0,1), as a string "0 or 1", as a Boolean (true or false). The code snippet of Set button is given here


Event: onClick

Function: Custom Code

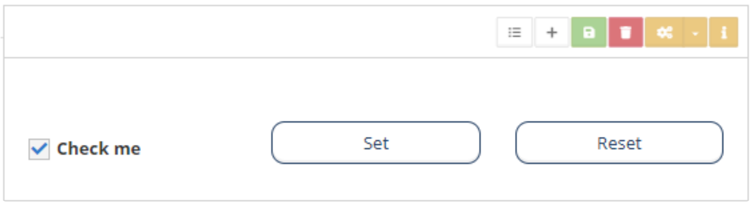
☐ Block input during event execution?

☐ Validate Input before execution?

Code

```
app.setFieldValue("cbCheckBox", 1);
//app.setFieldValue("cbCheckBox", "1");
//app.setFieldValue("cbCheckBox", true);
```

When set button is pressed, it sets the checkbox. Similar is the case when the reset button is clicked it will uncheck the checkbox. After clicking set button our app will look like this



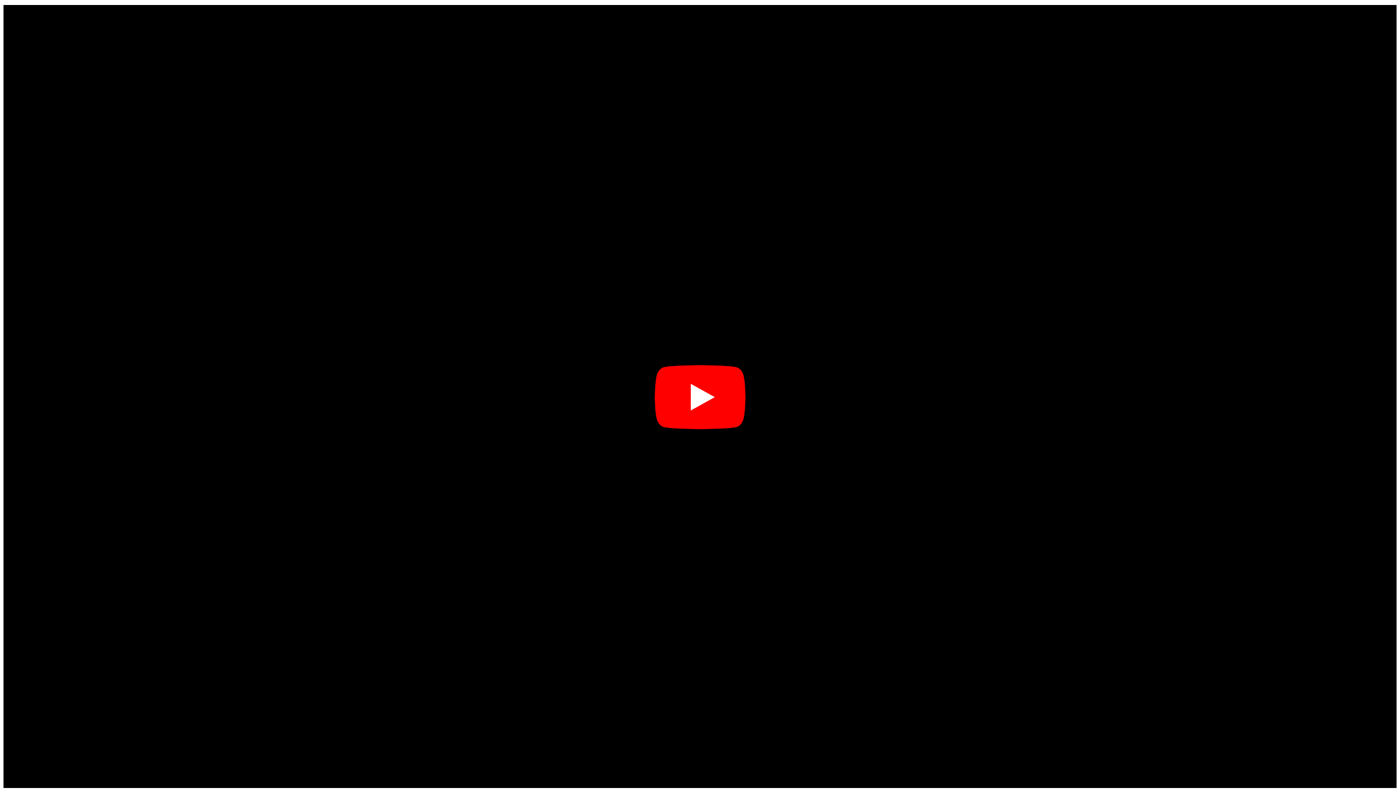
In the last, there is one point you need to care about whenever you want to use true and false to set or reset the checkbox do it in these three ways 0, "0" or false. Do not use true or false as string values like "false" while setting the value of checkbox. It will result in unexpected behavior.

ComboBox

ComboBox

Comboboxes are often used to refer to header records (an order line record will most likely have a combobox with its order record).

Demo



Documentation

In Brixxbox app configurations you can add different controls. One of them is ComboBox. Each control has four types of settings

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are main behavioral properties of any control. These are responsible for control's functionality.

- Control Type

- For using combobox control, combobox control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. By default Brixxbox assigns a random id which starts with underscore and followed by a string. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters. Each control is always accessed with its Control Id. For Example, you can get and set control value by using its ControlId. If you want to store value of this control in database, use the database button to create a column. You will be able to select column datatype, max length, can be null, options.

- Label

- It is the display value for control.

- Data

- This property specifies from where this control gets data. If no option is specified then by default "Not in Database" option is chosen. If you want to set it then you have to specify the controlId of data source.

- Refers to config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Combobox Key Field

- This is the field value from the sub data source. It also becomes the value of our combo box. When we have to get the value of our combo box using "getfieldvalue" function. It will return the corresponding id of selected combobox entry.

- Combobox Value Field

- This option defines how each entry of your combobox will be displayed. It can be a single or multiple columns from sub data source. For single column just name the column. For Example, "adrName". If you want to choose multiple columns, then use curly brackets. You can choose any place holders between them. For example, {id} - {adrName}. "-" is a place holder while id, adrName are column names.

- Min Search Length

- This option specifies the minimum search length of string to trigger search function. If you do not want any search function then specify "-1". In case you want to specify after how many characters Brixxbox should trigger the search and get actual records from the sub data source, specify it here as positive integer. For Example, if value 3 is specified then Brixxbox will trigger actual search only after 3 characters are entered in search field.

- Multiselect

- When this option is specified, one can select multiple values as combobox entries.

- Select List Button

- By default this is hidden. It opens a list of all items of sub data source. It can be used to select an entry from combobox.

- Select Edit Button

- By default this is enabled. If a value is selected in combobox then by clicking edit button will open the complete record in corresponding application. If it is clicked without any value it will open the corresponding application with no record. It can be used to add a new record.

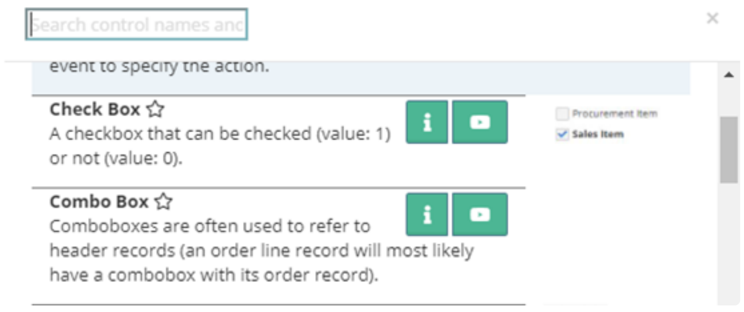
- Default Value

- This value is set when an application is opened or a new record is created.

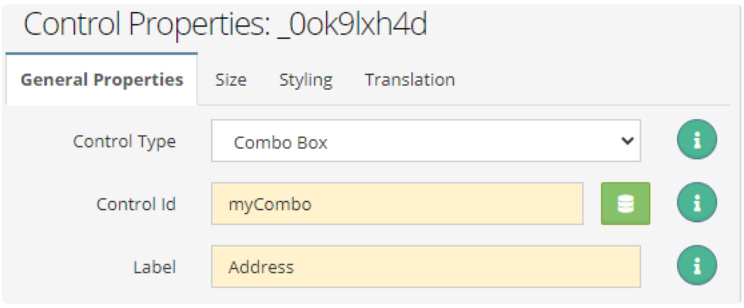
Tutorial

This tutorial will follow the similar example used in the demo for combobox. You can find this demo at the top of this page.

This tutorial assumes that a user has an existing app and they want to add a new control of type "Combo Box". First of all, select "combo box" from list of controls.

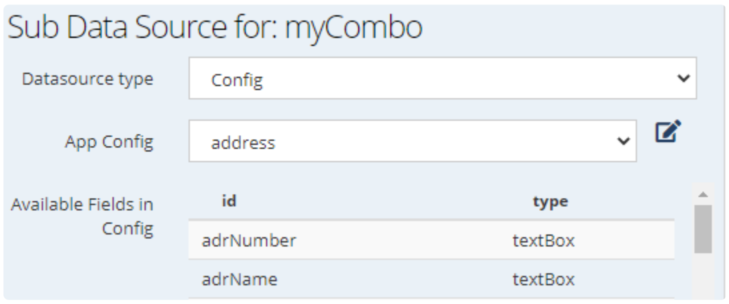

Search control names and
event to specify the action.
Check Box
A checkbox that can be checked (value: 1) or not (value: 0).
Procurement Item
Sales Item
Combo Box
Comboboxes are often used to refer to header records (an order line record will most likely have a combobox with its order record).

After that you need to give a id to your combobox and create column in database if necessary. Here we named it "myCombo". Also, add a public label for combobox. We labelled it "Address".



Control Type	Combo Box
Control Id	myCombo
Label	Address

Now save the control. It is time to assign a sub data source to our control. Click a sub data source button. Here you can assign different types of data sources. For this example, we will use config data source type. After that select the corresponding app config. We are selecting "address" config because we want to display all the addresses.

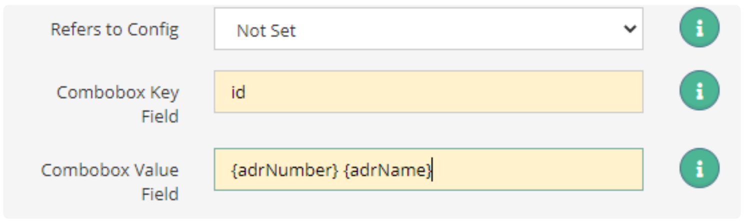


Datasource type	Config
App Config	address

Available Fields in Config

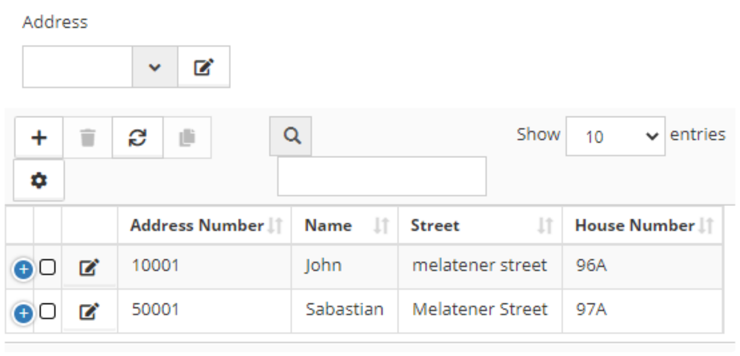
id	type
adrNumber	textBox
adrName	textBox

Up till now we have defined a combobox, assigned a unique control id and also added a sub data source to it. Now we want to assign two main properties to combobox: combobox key field, combobox value field. "Combobox key" property is a column from result set that will also be the value of our combobox when the entry of combobox is set. Mostly it is the id of result set. Here we also choose id. We can use this value for comparisons. For combobox value field, we want to display values of two columns: adrNumber, adrName from result set. We have to use curly brackets when using multiple columns.



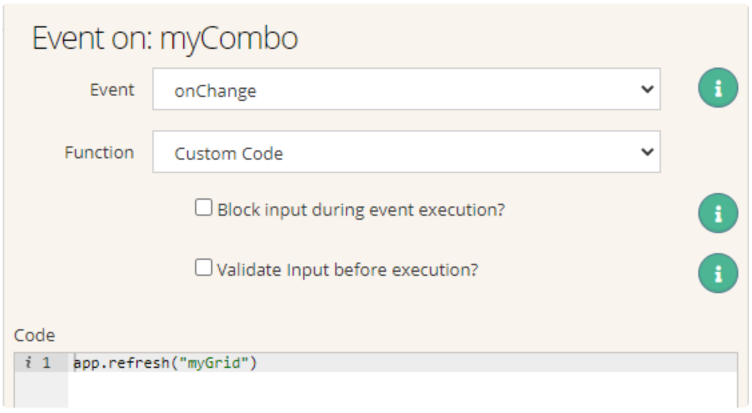
Refers to Config	Not Set
Combobox Key Field	id
Combobox Value Field	{adrNumber} {adrName}

Each entry of combobox will now represent the number of the address and the person's name. For example "10001 John". In this way we can define and use combobox but can we use the value of combobox to display or change some other data? the answer is yes!. For demonstration, we will add grid control (see grid documentation) "myGrid". We also assign address config as a sub data source for our grid control. Now "myGrid" control will display all the addresses present in the result set.



			Address Number	Name	Street	House Number
			10001	John	melatener street	96A
			50001	Sabastian	Melatener Street	97A

As you can see from above picture that nothing is selected in our combobox "Address". Now we want that whenever we select a record in combobox, the grid displays the same selected record only. For this purpose, we need to modify sub data source of mygrid and add a where clause. An important thing to note here is that all controls of an app are available as sql parameters. In where clause, we need to specify a condition which displays only a record which matches combobox value. We can specify this as "id=@myCombo". Remember that we selected "combobox key field" as id of sub data source "address". In order to pass this change to grid, we have to add a control event "onChange" to our combobox control. Whenever our combobox value change, we want "myGrid" to refresh.


Event on: myCombo

Event: onChange

Function: Custom Code

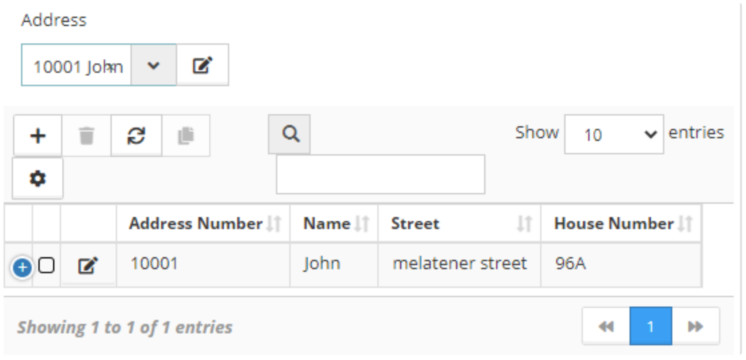
Block input during event execution? [ ]

Validate Input before execution? [ ]

Code

```
app.refresh("myGrid")
```

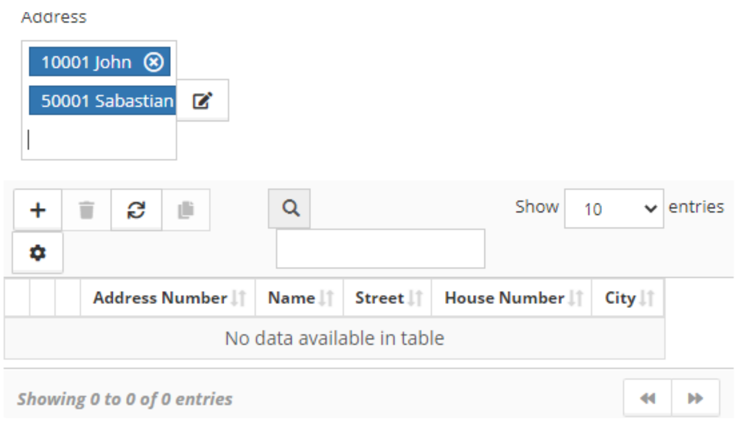
Now save and set a combobox value and we will be able to see only selected record in "myGrid". As we can see from figure below.



			Address Number	Name	Street	House Number
			10001	John	melatener street	96A

Showing 1 to 1 of 1 entries

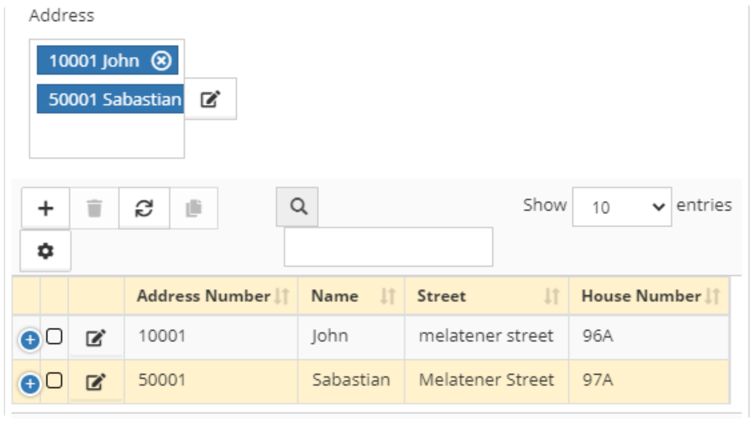
Combobox also allows us to select multiple records at a same time. To achieve this functionality, select "multiselect" checkbox from combobox properties. You can see from figure below that multiple values are selected in combobox but there is no record shown in grid.



		Address Number	Name	Street	House Number	City
No data available in table						

Showing 0 to 0 of 0 entries

This is because we were using where clause for comparing one value of combobox but now our combobox has multiple values. Now if we call "getfieldvalue" function on our combobox. It will return an array with selected combobox key value's. Now we have to update our where clause logic in grid to display all selected records. We add this statement to where clause **"id in (select value from OpenJson(@myCombo))"**. This statement says that if a value is present in result set then display it in grid.



			Address Number	Name	Street	House Number
			10001	John	melatener street	96A
			50001	Sabastian	Melatener Street	97A

DateBox

A date input control

In Brixxbox, you can add date box control to input date. Date box control allows you to enter dates through different methods. User can enter a date manually in text form or user can choose date from a calendar.

Documentation

Brixxbox app configurations allows you to add different controls in your app. Date box control allows user to add dates in different formats. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using Date Box control, "Date Box" control type should be selected from drop down list.

- Control Version

- In Brixxbox, there are 2 Versions of the Date picker available:

1: Version 1(V1) will display a HTML5 Browser build in Date Picker. This Control might look different in some browsers. 2: Version 2(V2) is a default version since since 10.8.2021. Now it will display a date picker from a library which is the same on all browsers and provides a variety of different methods to enter date.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. For Dates, it makes sense to label them via action that is being performed plus 'date' For example, in an order application one date should be delivery date. It will represent the action of delivering the order on a specific date to customer.

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Default Value

- Brixxbox allows users to add a default value for a control. This value will be set as a control's value when an application containing this control is opened or a new record is created. In the background, the brixxApi function "setFieldValue" is called with this value. For Date Box Control, we can use "moment()" function. It will assign the current value of date to our control.

Size

In Brixxbox, the screen is divided into 12 logical columns horizontally. For each control you can select explicitly how large it should be according to your application needs. Size properties defines how control will be displayed in Brixxbox App.

- Label on Top of Control

- If this option is selected then the label of a control and actual control will be in two different horizontal lines and control's label will be present on the top line and Date box control itself will be present on bottom line. If this option is not selected then 12 column's will be shared between label and control itself and both of them will be present on same line in side by side formation. For example, label can be 4/12 column's long and control can be 8/12 columns long over same horizontal line.

The display layout is divided into four different categories depending upon the pixels. For each category, Brixxbox allows you to customize the width of control.

- Phone (and up)

- This category includes devices which have screen pixels less than 768 pixels. For example: Smart phones. This will be by default selected category in Brixxbox. In this category, each control will be assigned to 12/12 by default. It means that in extra small screens, each control takes up the whole horizontal line and next control will take up whole next horizontal line and soon.

- Tablets (and up)

- This category includes devices which have screen pixels less than 992 pixels. For example: Tablets. By switching this settings on, they will come into play for particular app.

- Medium Devices (Laptops)

- This category includes devices which have screen pixels less than 1200 pixels. For example: Laptops. In Medium devices, an input control will be set to 4/12 per control by default with option to customize the size of controls.

- Large Devices (Big Monitors)

- This category includes devices which have screen pixels greater than 1200 pixels. For example: Big monitor displays.

- Custom Label class

- This is only relevant if the control is in a From Group, and so the Label is left or right of the control instead of above it. It uses the same rules as the Width properties as above.

Styling

These properties defines how control looks like in Brixxbox.

- Enabled

- It allows two options: yes, no. If "yes" is selected, the control will be enabled. In Date Box case, user will be able to select a date. On the other hand, if "no" is selected then it will be disabled.

- Disabled in edit mode

- There are two modes available in Brixxbox for each app: edit mode, create mode. In edit mode: user is allowed to add new controls and functionality while in create mode, on the basis of app controls, user is allowed to add a new record, edit an existing record, or list all the records. In order to edit a record, user has to list all the records using app settings and then select a specific record to edit. "Disabled in edit mode" is a checkbox field. If this option is selected then the control will not be functional while editing a previously saved record but it will be functional while creating a new record. In "Date time box" case, user will not be able to edit the value of this control f previoudded record.

- Visibility

- It will specify the visibility options available in Brixxbox. There are four options available: visible, hidden, hidden in grids, and hidden in forms.

- Text Color

- Brixxbox allows user to specify, text colors of controls. There are seven different options available in Brixxbox.

- Input Size

- With this option, Brixxbox enables users to set the input size of Date Box Control. There are four input sizes available for Date Box Control: Default, Extra Small, Small and Large.

- Tooltip

- Enter a text for a tool tip. This will be translated.

- Help Text

- Brixxbox with the property allow users to add some help text. This text will be displayed below the control. The purpose of this option is to encourage the right behavior of customer while using a specific control.

For further information please have a look at Bootstrap Grid System

Tutorial

This tutorial assumes that a user has an existing app and they want to add a new control of type "Date Box". In this app, we have two textboxes "first name", "last name" and a "save" button. "Save" button validates the inputs and save them, when button is pressed. Now we also want to store joining date of employees in our app. To allow this behavior we need to add a "Date Box" control.

First of all, select "Date Box" from list of controls. In second step, select the version of Date Box Control i.e. "V1" or "V2". Here we selected "V2". After this, you need to give an id to your Date Box. Next assign "Joining Date" label to your control and add "nonEmpty" validator. Now save your application. Your application should look like this:

Now I you always want current date as a value of "Joining Date" control, we can use "moment()" function as a default value for our control in general properties. It will assign the current date to new records and on creating new records, our App will look like this

The value of joining date is "08/30/2021". It shows the current date at which I am writing this tutorial. Date Box allows you to add date in text form. For example, if your employee joinied on 1st of August 2021 then you will add this in text form and in this format "mm/dd/yyyy". It will become 08/01/2021. Date Box allows you to add input in other format also. For this you need to click on calendar icon. IT will display the calendar as shown below

Button on the top left of calendar acts same as "moment()" function. By clicking it one can set current date as a date picker's value. Button on top middle of calendar is used to reset date picker's value. Cross button(X) is used to exit to calendar view. From calendar we can set year, month and day easily with mouse clicks without typing date.

DateTimeBox

DateTimeBox

In Brixxbox, you can add date time box control to input date with timestamp. Date time box control is similar to Date Time Box control in Brixxbox, it also allows users to add date's with an addition of time stamp. User can select date and time from calendar view.

Documentation

Brixxbox app configurations allows you to add different controls in your app. Date time box control allows user to add dates plus time. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using Date Time Box control, "Date Time Box" control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. For Dates and time control, it makes sense to label them via action that is being performed plus 'time' For example, in an employee information application one date should be arrival time which store date plus time everyday.

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Default Value

- Brixxbox allows users to add a default value for a control. This value will be set as a control's value when an application containing this control is opened or a new record is created. In the background, the brixxApi function "setFieldValue" is called with this value. For Date Time Box Control, we can use "moment()" function. It will assign the current value of date to our control.

Date and Time Box Properties

- Precision

- This property allows users to select timestamp precision. It has two options: minutes and seconds. By default minutes is selected.

- Save UTC and show local time

- This is checkbox option. If this Checkbox is selected. The control will display date and time in local time zone of user but the orgnal value behind the control is utc timestamp. Brixxbox "setFieldValue" function will assume you set a utc time and will add the local timezone just for the display. Brixxbox "getFieldValue " function will give you the utc time as well. So users in different timezones will see different values, but in the database there is just the utc time stored.

Size

In Brixxbox, the screen is divided into 12 logical columns horizontally. For each control you can select explicitly how big it should be according to your application needs. Size properties defines how control will be displayed in Brixxbox App.

- Label on Top of Control

- If this option is selected then the label of a control and actual control will be in two different horizontal lines . Control's label will be present on the top line and Date time box control itself will be present on bottom line. If this option is not selected then 12 column's will be shared between label and control itself and both of them will be present on same line in side by side formation. For example, label can be 4/12 column's long and control can be 8/12 columns long over same horizontal line.

The display layout is divided into four different categories depending upon the pixels. For each category, Brixxbox allows you to customize the width of control.

- Phone (and up)

- This category includes devices which have screen pixels less than 768 pixels. For example: Smart phones. This will be by default selected category in Brixxbox. In this category, each control will be assigned to 12/12 by default. It means that in extra small screens, each control takes up the whole horizontal line and next control will take up whole next horizontal line and soon.

- Tablets (and up)

- This category includes devices which have screen pixels less than 992 pixels. For example: Tablets. By switching this settings on, they will come into play for particular app.

- Medium Devices (Laptops)

- This category includes devices which have screen pixels less than 1200 pixels. For example: Laptops. In Medium devices, an input control will be set to 4/12 per control by default with option to customize the size of controls.

- Large Devices (Big Monitors)

- This category includes devices which have screen pixels greater than 1200 pixels. For example: Big monitor displays.

- Custom Label class

- This is only relevant if the control is in a From Group, and so the Label is left or right of the control instead of above it. It uses the same rules as the Width properties as above.

Styling

These properties defines how control looks like in Brixxbox.

- Enabled

- It allows two options: yes, no. If "yes" is selected, the control will be enabled. In Date Time Box case, user will be able to select a date and time. On the other hand, if "no" is selected then it will be disabled.

- Disabled in edit mode

- There are two modes available in Brixxbox for each app: edit mode, create mode. In edit mode: user is allowed to add new controls and functionality while in create mode, on the basis of app controls, user is allowed to add a new record, edit an existing record, or list all the records. In order to edit a record, user has to list all the records using app settings and then select a specific record to edit. "Disabled in edit mode" is a checkbox field. If this option is selected then the control will not be functional while editing a previously saved record but it will be functional while creating a new record.

- Visibility

- It will specify the visibility options available in Brixxbox. There are four options available: visible, hidden, hidden in grids, and hidden in forms.

- Text Color

- Brixxbox allows user to specify, text colors of controls. There are seven different options available in Brixxbox.

- Tooltip

- Enter a text for a tool tip. This will be translated.

- Help Text

- Brixxbox with the property allow users to add some help text. This text will be displayed below the control. The purpose of this option is to encourage the right behavior of customer while using a specific control.

For further information please have a look at Bootstrap Grid System

Tutorial

This tutorial assumes that a user has an existing app and they want to add a new control of type "Date Time Box". In this app, we have two textboxes "first name", "last name" and a "save" button. "Save" button validates the inputs and save them, when button is pressed. Now we also want to store arrival datetime of employees in our app. To allow this behavior we need to add a "Date Time Box" control.

First of all, select "Date Time Box" from list of controls. After this, you need to give an id to your Date Time Box. Next assign "Arrival Time" label to your control and add "nonEmpty" validator. Now save your application. Your application should look like this:

Now if you always want current date as a value of "Arrival Time" control, we can use "moment()" function as a default value for our control, in general properties. It will assign the current date time to new records and on creating new records, our App will look like this

The value of Arrival Time is "08/31/2021 4:05 PM". It shows the current date and time at which I am writing this tutorial. Date Time Box allows you to add date time in calendar format. For this you need to click on calendar icon. It will display the calendar as shown below

Button on the top left of calendar acts same as "moment()" function. By clicking it one can set current date and time as a control's value. Button on top middle of calendar is used to reset date picker's value. Cross button(X) is used to exit to calendar view. From calendar we can set year, month, day and time easily with mouse clicks without typing date and time.

DocViewer

The Doc Viewer can be used to show blobs or create reports like the report control.

FileImport

The file import Control allows users to upload a file.

The control will trigger onFileImport

FormGroup

A form group is another structural control in Brixxbox controls. It is a structural element to create apps with side by side columns.

FormGroup

Normally Brixxbox takes the responsibility of placing control in an app. In order to group element in a row we can use row control. For a vertical alignment, you can place Controls inside the formgroup or you can also add a FormGroupRow in which you can add controls horizontally inside formgroup.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "FormGroup" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will see how to use formgroup control to achieve vertical grouping of controls. This control is purely for structuring controls. For this tutorial, we will be using item app. Our app has 5 controls: item id(numbox), | item name(text box), sale price(numbox), quantity(numbox), and quantity unit type(combobox). Right now, the order of controls is selected by us but the placement is selected by Brixxbox. Our app looks like this:

Now we want to present item id and item name in a vertical fashion. For that, we need formgroup. Lets add a formgroup from the controls list and add both item id and item name as its children's. Now these two controls with be placed vertically and Brixxbox will try to place other controls in horizontal fashion, wherever it finds a place. Our app now looks like this:

As we can see in the above snapshot that item id and item name controls are placed vertically and other controls are placed row wise. The yellow area corresponds to the area assigned to formgroup. If there is enough space available after formgroup, brixxbox will start placing other controls there. Now we also want to add another formgroup for all other controls. Lets add a new control of formgroup type and place remaining controls as its children:

Now we have two vertical columns of controls because we are using two formgroups. Another thing to note here is the size of each formgroup is 4/12 of complete row. I want to add another formgroup and put quantity type control in it. Why I want to do that? because I want my app to be limited to two rows:

Now compare the final version of app with start, it is more organized.

FormGroupRow

Brixxbox provides FormGroupRow control in order to place control in horizontal fashion inside a FormGroup

FormGroupRow

We can use this formgrouprow control to make a horizontal row of controls inside a fromgroup specifically. It serves the same purpose as of row, make horizontal row of controls.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "FormGroupRow" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will see how to formgroup control to achieve vertical grouping of controls. This control is purely for structuring controls. For this tutorial, we will be using item app. Our has 5 controls: item id(numbox), | item name(text box), sale price(numbox), quantity(numbox), and quantity unit type(combobox). We also updated our app in "formgroup" tutorial, added the formgroups to place controls vertically, and now looks our updated app looks like this:

Now we want to place quantity and quantity unit type in a one line. For that, we can use formgrouprow. Lets add a form group row from the controls list and add it as a child to third formgroup. After that place both iquantity and quantity unit type as formgrouprow childern. Now these two controls with be placed horizontally and Brixxbox will place them in horizontal fashion. Our app looks like this:

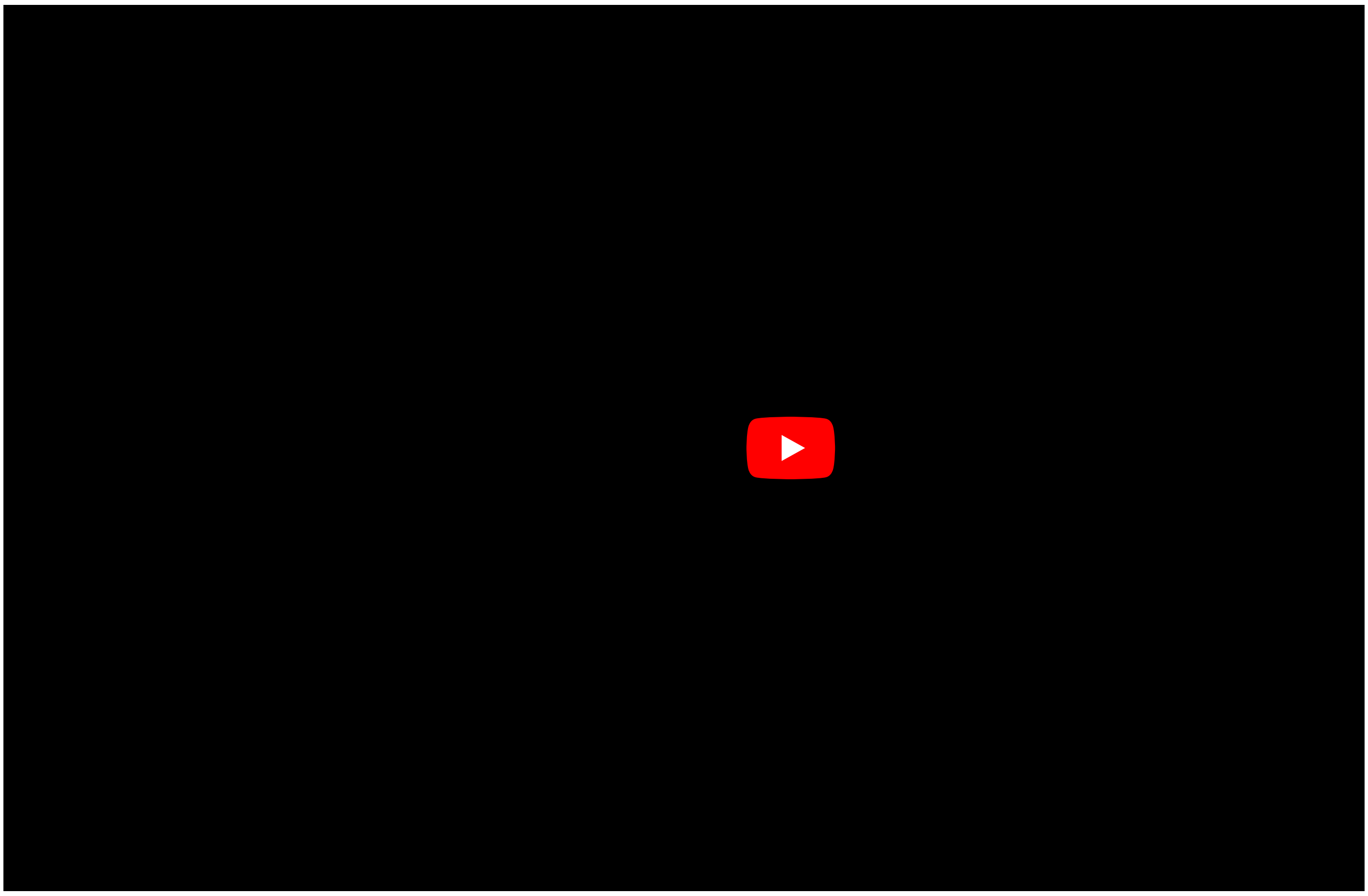
As we can see in the above snapshot that quantity controls are placed in a horizontal line but they don't look very nice. The reason is there are cluttered with other controls. Lets use our knowledge of Horizontal Line which is a styling control. This will divide our third form group from others. Lets add this to our app:

FullCalendar

Calendar

This Control provides an outlook like calendar.

Demo



Usage examples

Setting a date to a calendar control scrolls to that date.

```
brixxApi.setFieldValue("myCalendarControl", "2019-01-31");
```

Sub Settings

The Calendar supports sub settings to set. You can add resource(s) to the calendar for example:

Resources

You can add resources to your calendar by using the setFieldValue comand. To see and use them, you have to enable to resource views in the contr

```
//Resources will delete all and set a new array of resources
brixxApi.setFieldValue("myCalendarControl.resources", [
{
    id: 1,
    title: "Room A",
},
{
    id: 2,
    title: "Room B",
}]);

//resource will add a single resource
brixxApi.setFieldValue("myCalendarControl.resource", {
    id: 1,
    title: "Room A",
    mySortCrit: 100
});
```

Resources can be grouped. You need a grouping criteria in your resources and you must tell the calendar the name of this criteria. The order of both
matter.

```
//Set the grouping criteria
brixxApi.setFieldValue("myCalendarControl.resourceGroupField", "building");

//Add the resources
brixxApi.setFieldValue("myCalendarControl.resources", [
   {
       id: 1,
       building "Main building",
       title: "Room A",
   },
   {
       id: 2,
       building "Main building",
       title: "Room B",
   },
   {
       id: 3,
       building "Side building",
       title: "Room C",
   },
   {
       id: 4,
       building "Side building",
       title: "Room D",
   }
]);
```

The width of the resource side panel can be set as well. Can be specified as a number of pixels, or a CSS string value, like "25%".

```
brixxApi.setFieldValue("myCalendarControl.resourceAreaWidth", "30%")
```

You can sort the resourceList by a setFiedlValue command as well.

```
brixxApi.setFieldValue("myCalendarControl.resourceOrder", "title");
brixxApi.setFieldValue("myCalendarControl.resourceOrder", "-title"); //sort descending
brixxApi.setFieldValue("myCalendarControl.resourceOrder", "mySortCrit,title"); //sort by your custom resource property mySort
```

To assing events to resources, set the resourceId property in the DataTransform event like the start or end time.

```
    eventArgs.details.title = eventArgs.details.myTitle;
    eventArgs.details.start = eventArgs.details.myStart;
    eventArgs.details.end = eventArgs.details.myEnd;
    eventArgs.details.resourceId = eventArgs.details.myRoomId; //set resourceId to assing the event to a resource
```

Views

You can switch to a specific view mode by calling a set value. The following view modes are available, but might be restricted to your control settings

- dayGridMonth
- timeGridWeek
- timeGridDay
- listWeek
- resourceTimeline
- resourceTimelineWeek
- resourceTimeGridDay

```
      brixxApi.setFieldValue("myCalendarControl.view", "resourceTimeline");
```

Grid

Grid

A grid displays a list of records, specified by the sub data source of that control.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Grid" control type from drop down.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Hide Select Checkbox Column

- By default, grid also display's a checkbox with each row. This checkbox is used to select a specific row. If it is enabled then user can select multiple rows at a time and perform different operations on them like deletion. If user does not want to display this checkbox then this property should be checked.

- Hide Edit Button Column

- This property is by default unchecked. It allows grid to display an edit button with each displayed row. If user does not want to display Edit button then this property should be checked.

- Cascade Delete

- This is a checkbox. If it is set and a delete operation is performed on a record, in an app containing this Grid control then all the lines of a record will get deleted. For example: User wants to delete an order with all of its item's lines.

- Cascade Copy

- If it is set and a copy operation is performed on a record, in an app containing this Grid control then all the lines of a record will get copied. For example: User wants to copy an order with all of its item's lines.

- Server Side Paging

- Use this property for big data tables. If checked, grid will only be able to data from server side only instead of getting all the data from data source.

- Smooth Scroll

- If checked, this property allows the users to scroll through list of items of grid instead of scrolling up and down via paging buttons.

- No automatic refresh

- Use this property when fetching expensive Grid data. If set, it will block unnecessary calls for data. Brixxbox allows an app to refresh data of all of its controls using "app.refresh()" function. Settings this property will exclude this grid from data refresh calls. However, you can refresh grid data by specifying the grids name in refresh function. For Example: app.refresh("myGrid").

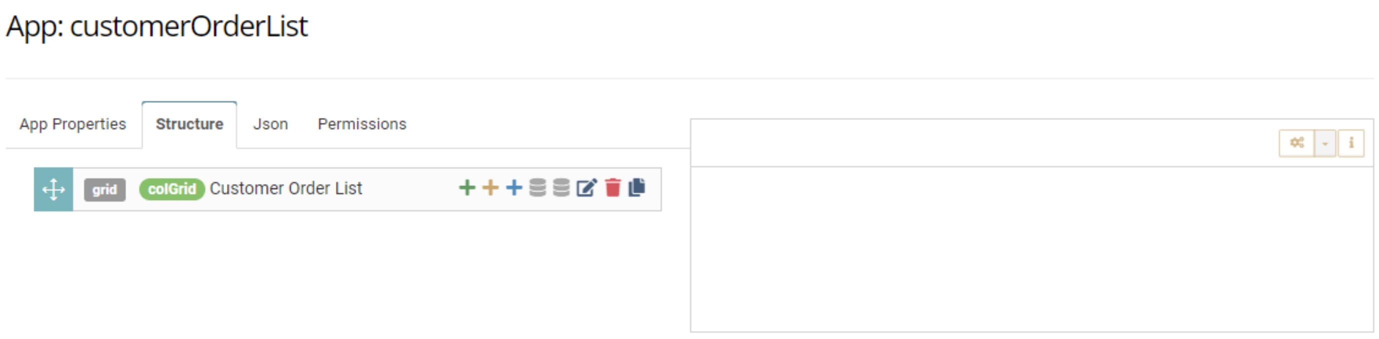
Tutorial

Gird control can be used in two possible scenarios: displaying all data of sub data source. For example: listing all customer orders, and displaying some values. For example: listing order lines of a specific customer order only.

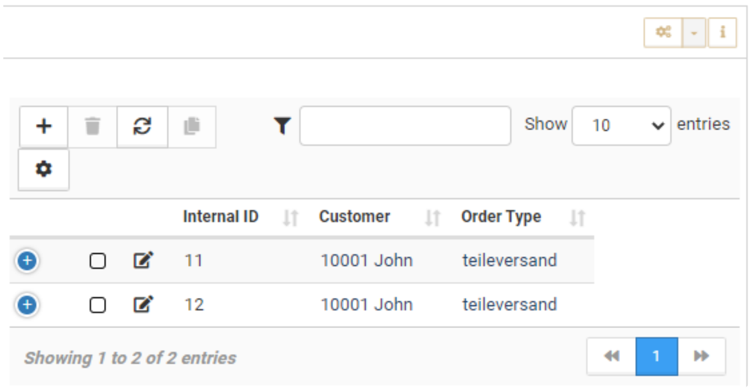
This tutorial assumes that a user has an existing customer order app which records customer's order data. They want to list all customer orders in a new app and update grid to display all order lines of a specific customer order in customer order app.

For first scenario, create a customer order list app, select "Grid" from list of controls. After that you need to give an id to your grid. As grid is usually used to display data from a sub data source, so there is no need to create column in database for Grid control.

Let start with the first case. Select a new app, name it "customerOrderList". Select a "Grid" control type from the list, assign a unique id and meaningful label:


App: customerOrderList

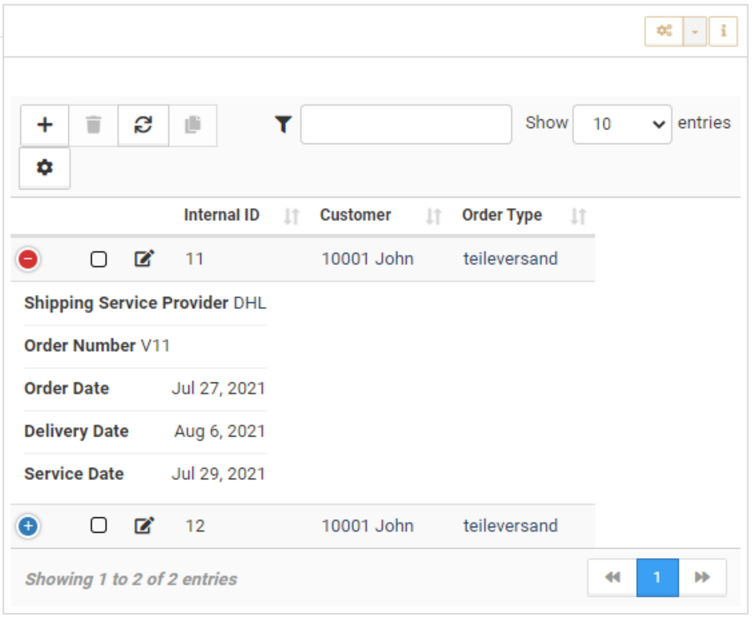
As you can see from the above picture, we have added a grid control but nothing appears in app editor on right side. This is not an error but expected behavior because grid control is used to display a list of records but first we need to specify "sub data source" in grid properties. For this click on "edit sub data source" and select "config" as a data source type. Now select "CustomerOrder" app to display all customer orders and save app settings.



			Internal ID	Customer	Order Type
			11	10001 John	teileversand
			12	10001 John	teileversand

Showing 1 to 2 of 2 entries

Now you can see that our grid list all customer orders. In its menu, grid allows multiple options like adding new data row, deleting data row, refresh grid, copy data, search filter and drop down list for specifying number of entries for display. Grid provides each data row with plus, select, and edit button. Depending on screen size grid displays only few data columns and plus button is used to see remaining columns as shown below:



Show 10 entries

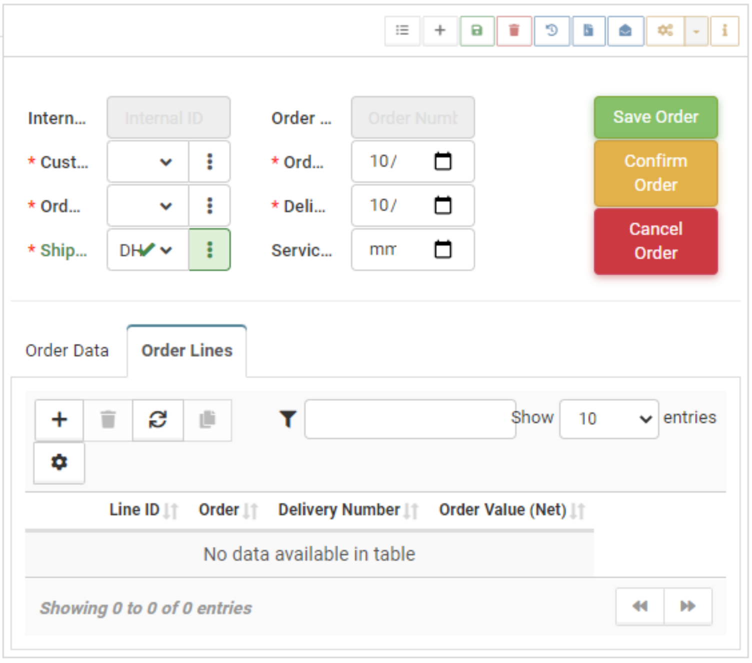
			Internal ID	Customer	Order Type
⊖	☐	✎	11	10001 John	teileversand

Shipping Service Provider	DHL
Order Number	V11
Order Date	Jul 27, 2021
Delivery Date	Aug 6, 2021
Service Date	Jul 29, 2021

			Internal ID	Customer	Order Type
⊕	☐	✎	12	10001 John	teileversand

Showing 1 to 2 of 2 entries

Grid allows us to enable or disable select and edit buttons(see General Properties). We have successfully listed all customer orders, now we move towards second scenario. We have two relations between customer order and order lines which are: customer order can have multiple order lines, and one order line can only belong to one customer order. In this part, we want to cover second relation. Now we want to display order lines which belong to only one customer order in customer order app grid. Our customer order app looks like this:



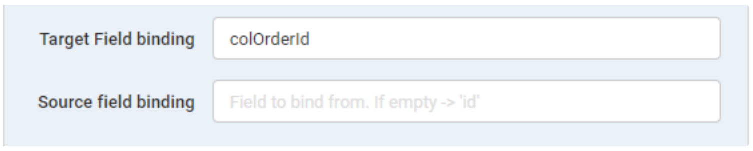
Intern...	Internal ID	Order ...	Order Numl
* Cust...		* Ord...	10/
* Ord...		* Deli...	10/
* Ship...	DH	Servic...	mm

Show 10 entries

Line ID	Order	Delivery Number	Order Value (Net)
No data available in table			

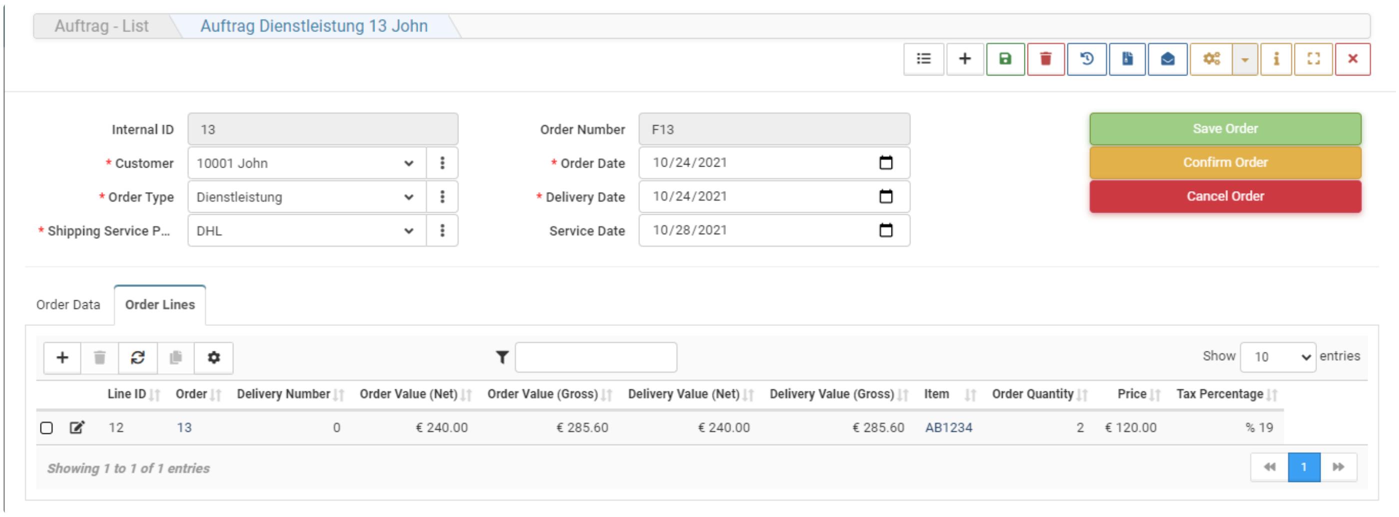
Showing 0 to 0 of 0 entries

There are a lot of details in this app which are not important here. Our concern is order lines gird which will display only those order lines which belong to one specific order. To allow this behavior, we need to edit "sub data source" settings of our order lines grid and set "target field binding" with the customer order line id.



Target Field binding	colOrderId
Source field binding	Field to bind from. If empty -> 'id'

Now we will see only those order lines in our grid which belong to same customer order. Lets add a new order for our customer John and add a new order line for this order.



Auftrag - List | Auftrag Dienstleistung 13 John

Internal ID	13	Order Number	F13
* Customer	10001 John	* Order Date	10/24/2021
* Order Type	Dienstleistung	* Delivery Date	10/24/2021
* Shipping Service P...	DHL	Service Date	10/28/2021

Save Order · Confirm Order · Cancel Order

Order Data | Order Lines

Show 10 entries

Line ID	Order	Delivery Number	Order Value (Net)	Order Value (Gross)	Delivery Value (Net)	Delivery Value (Gross)	Item	Order Quantity	Price	Tax Percentage
12	13	0	€ 240.00	€ 285.60	€ 240.00	€ 285.60	AB1234	2	€ 120.00	% 19

Showing 1 to 1 of 1 entries

Now order lines which belong to John's order are being displayed in our order lines grid. In our case we only added one order line.

GridConfig

A GridConfig can be places as a child under a grid to specify the columns. This is useful, when the grid is based on a SQL Statement rather than a config.

GroupBox

A GroupBox is another structural control in Brixxbox controls. It is a structural element to create apps with side by side columns just like formgroup control

GroupBox

Normally Brixxbox takes the responsibility of placing controls in an app. In order to group controls in a row we can use row control. For a vertical alignment, you can formgroup or groupBox. Groupbox is the advanced form of formgroup. It has all the functionalities of formgroup. Additionally, it also has a header and a border.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "GroupBox" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will see how to use groupbox control to achieve vertical grouping of controls. This control is purely used for structuring controls. For this tutorial, we will be using item app. Our app has 5 controls: item id(numbox), | item name(text box), sale price(numbox), quantity(numbox), and quantity unit type(combobox). Right now, the order of controls is selected by us but the placement is selected by Brixxbox. Our app looks like this:

Now we want to present item id and item name in a vertical fashion. For that, we need Groupbox. Lets add a Groupboxfrom the controls list, name it "item details" and add both item id and item name as its children's. Now these two controls with be placed vertically and Brixxbox will try to place other controls in horizontal fashion, wherever it finds a place. Our app now looks like this:

As we can see in the above snapshot that item id and item name controls are placed vertically and groupbox containing them has a clear boundary and a header while other controls are placed row wise. If there is enough space available after groupbox, brixxbox will start placing other controls there. Now we also want to add two more groupboxes for other controls. Lets add two new groupbox type controls and place quantity controls in one groupbox and price control in second one as their children respectively:

We can see the final version of app is more organized. Item details, quantity details, and price details are in separate groupboxes with meaningful header and visible boundary makes our app much more readalbe.

HorizontalLine

HorizontalLine

Most of the controls are linked with storing and displaying information but horizontal line is purely styling control. It is used to group fields or to separate different controls. For example, separating two row controls. As the name suggests, it will be an horizontal line between controls.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App. In case of horizontal line, these properties are of less importance.

- Control Type

- For using grid control, click on "Add Control" and select "Horizontal line" control type from drop down list. As this most commonly used control, Brixxbox has a designated button for adding a new row. User can find this button on app editor page and on the top right corner.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will see how to use horizontal line control. this control is purely for styling. For this tutorial, we will be using "unitDemo" app. We have used this app in "unit" control tutorial also. If you want to look into details of this app please go through the "unit" documentation. Here is the link: | Row Documentation. Our app looks like this:

We have already separated controls with rows but there is no visible barrier between them. Now this is where "horizontal line" comes to help. Now we want to visually separate item id, item name and sale price, sale price test then we should put a horizontal line between them. To add horizontal line, click on controls list, select "horizontal line" control. It will be placed at the end of app tree. Drad this control and place it between two rows. Now our app will look like this:

This looks nice. Now lets also separate Unit type control from second row. We need to add another horizontal line between them. After that our app looks like this:

In this way, we can separate different group of controls visibly.

HtmlTable

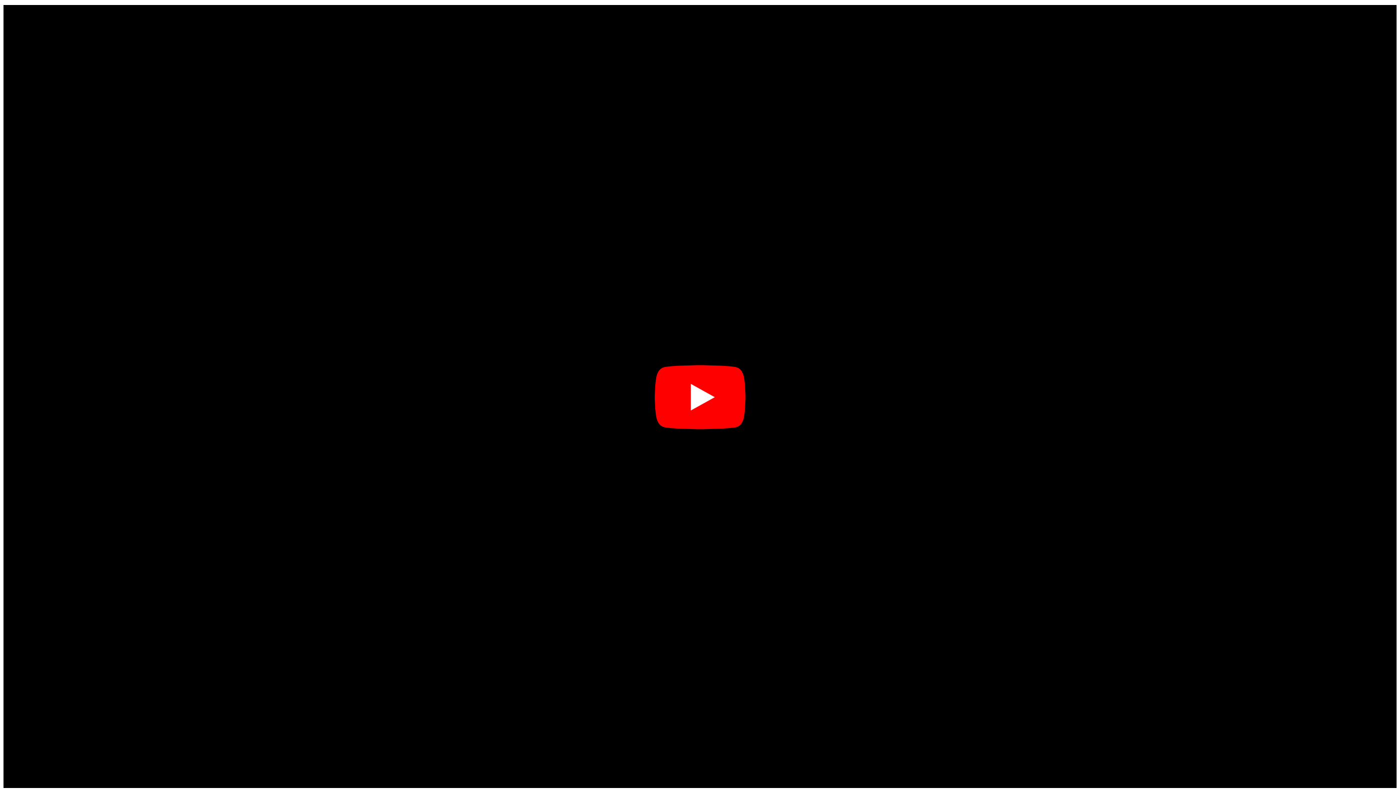
This Control provides an Html Table control, where you can custom specify the content.

Translation

If you want to translate parts of your table, like the column headers, you can use translate tags The part after the ':' in the curly brackets will be translated, like a label. This can be used in header, footer and body.

```
<th>{Translate:Address}</th>
<th>{Translate:First Name}</th>
<th>Id</th>
```

Demo



HtmlTemplate

You can add your custom html code with this control

Demo

Take a loot at Custom HTML Templates for examples

Image

This control displays an image. In edit mode it accepts an image url.

Label

A label control is designed to be a standalone text

It can be used as headings or giving specific instructions about usage of different app controls or display value for other controls. This control has text but without any interaction. It means that label text is not clickable.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Label" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will be using "customerOrderList" app. It is a simple app which consists of a grid and a button. The grid lists all available customer orders. The button "Number of Orders" does not have any "onClick" event because we need it only for displaying badge over it.

Here we can see the list of customer orders but there is no label specifying to whom this list represents. A new user will have to put some effort and explore about the grid entries. Our job is to make user's life easier. For this purpose we can add a label, which will label the list grid as "Customer Order List". We can also apply different color settings also to this label. Lets now add a label, select a "label" control from the control's list. Assign control id and label. We can also set default value for this label control. In this case default value property takes precedence over label property of this "label" control.

Now we have a nice label text showing what does this grid is displaying. Let just assume our user is new and he don't know how to edit a specific customer order then we can a label explaining to click edit button on specific customer order to open that order and edit it. Lets add a label for this and put as a child to our customer order list grid.

Now it is showing the user that in order to open a specific click on edit button.

MultilineTextBox

This Brixxbox control is similar to textbox control. Thedifference is that it is resize able multiline text box.

It does not provides any formatting features. In Brixxbox, if you need formatting then you should use WysiwygText (What you see is what you get).

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

There is an additional property type for multilinetextbox. They are related to maximum and minimum height a multilinetextbox can adapt to.

1. Multiple Line Text Input Properties

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "mulitLineTextBox" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Default value

- This value is set when an application is opened or a new record is created.

Multiple Line Text Input Properties

- Minimum height

- It is a drop down list. It is selected in the form of lines. Its value ranges from 2 to 15 lines or no minimum height.

- Maximum height

- Maximum height:

- It is a drop down list. It is selected in the form of lines. Its value ranges from 2 to 15 lines or no maximum height.

- Initially adjust height to content

- If its value is checked then whatever the height of initial data present will also become the height of Multi line text input.

Tutorial

In this tutorial, we will see how to use multilineTextBox control. For this tutorial, we will be using item app. Our app has 5 controls: item id(numbox), | item name(text box), sale price(numbox), quantity(numbox), and quantity unit type(combobox). These five controls are placed in three different groupbox controls: item details, Quantity Details, and Sales Details. If you want to dig deeper in this app. Please go through group box documentation. Our app at start looks like this:

Now we want to add item description in our app. This description can be very long so we do not want to use text box. For this purpose, we will use multilinetextinput. In the above snapshot, user can also notice that all other controls are in groupbox containers so lets also add a new groupbox control named "Item Description" and add a new multilinetextinput control and place it inside item description groupbox control.

Here you can see multi line text input control. Interesting thing is you can change size of this control both in editor mode and in app mode. If you check "Initially adjust height to content" property then whenever this app gets initialized and data is being loaded from a data source, it will take the height of the text present in it.

NumBox

A numeric Text box lets the user to enter a number. It can be used for saving numerical input and also for displaying numerical data. You can use various control validators to check validity of input like "nonEmpty", "digits", "lessThan", and "zipCode" etc. For example, if you want to store a zip code for addresses, you can use a num box for input and then use a control validator "zipCode" on it. "zipCode" validator allows users to set two properties: custom message, country code. User can set any message according to application needs and specify the country name. Brixxbox will display error if zip code entered by user is not valid with respect to country code provided in settings.

Documentation

Brixxbox app configurations allows you to add different controls in your app. Num box is the simplest control. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using num box control, "Numeric input" control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Select all on focus

- This property when checked allows to select all the data in the num box, whenever this num box comes in focus.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Decimal Digits

- This option allows you to set whole numbers or real numbers up to 10 decimal precision as a valid input for your num box.

- No Up/Down arrows

- If you check this option then no up/down arrows will be provided next to number. The default functionality of up/down arrows is to provide increment by 1 or decrement by 1 when upper arrow or down arrow is clicked respectively.

- Unit

- The property allows you to specify a unit character here. For example we can specify $ for US Dollar, € for Euro etc.

- Load record on load

- If this checkbox is selected then Brixxbox will load the record using value of this text box as a key.

- Default value

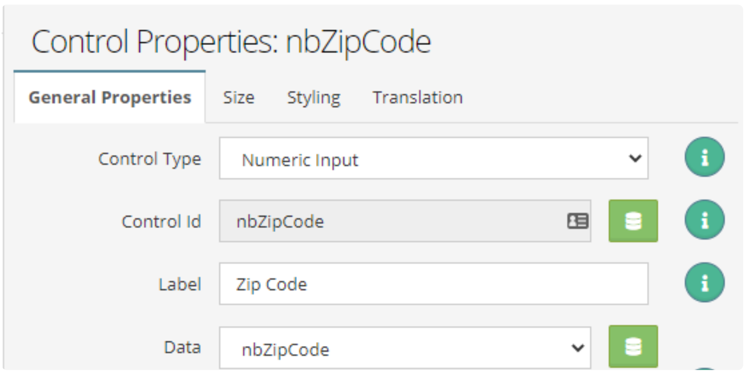
- This value is set when an application is opened or a new record is created.

Tutorial

This tutorial assumes that a user has an existing app and they want to add a new control of type "Numeric Input". First of all, select "Numeric Input" from list of controls. After that you need to give an id to your num box and create column in database if you want to store the input in database.

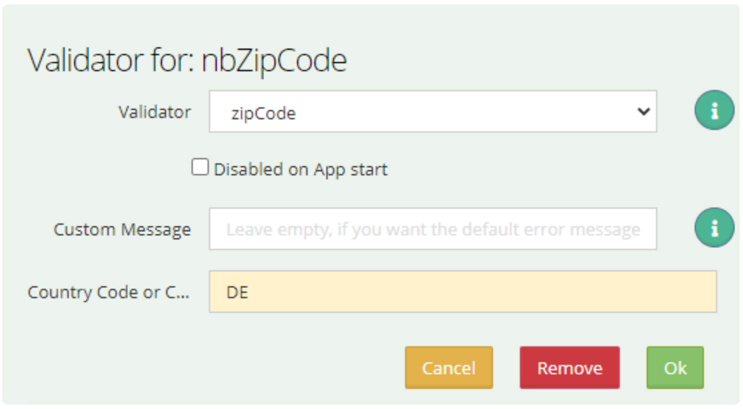
Here we will consider two possible scenarios: getting a value from user for our num box and saving it, and on the basis of other control displaying some value in our num box.

Let start with the first case. For our Tutorial, select a "Numeric input" control type. Assign a meaningful controlId to it. We name it "nbZipCode". Here we add "ZipCode" name with mandatory prefix of the app "nb". Add it to database because we want to store zip codes. We have also assigned "Zip Code" label to it. Now save it.


Control Properties: nbZipCode

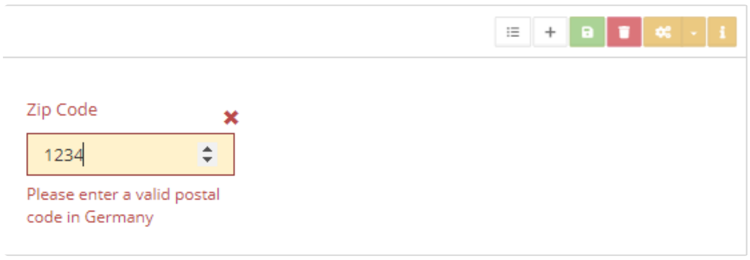
Control Type: Numeric Input
Control Id: nbZipCode
Label: Zip Code
Data: nbZipCode

With above settings, any numeric values can be accepted as a valid input but we want to change this behavior of num box. We want to save zip codes of Germany in our num box. The question arises, How can we alter the settings of num box so that it only accepts valid zip code? A valid German zip code consists of only 5 positive digits. To achieve it, we add a control validator "zipCode" to our num box. In its settings, for country code we choose "DE" for Germany and save it.

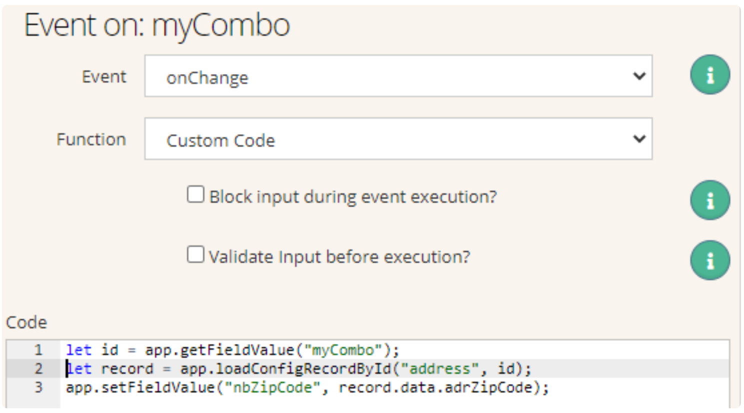

Validator for: nbZipCode

Validator: zipCode
Disabled on App start
Custom Message: Leave empty, if you want the default error message
Country Code or C...: DE

Now our num box only allow 5 digits positive numbers. Otherwise it will give an error.


Zip Code
1234
Please enter a valid postal code in Germany

Now we will take a look into second case. Let take the address comboBox which we used in the combo box tutorial. Now when we set an address in our combobox, we want to display its corresponding zipcode in our num box. We have to write a custom code on "onChange" event of combobox. This event will be triggered when combo box value change. The code looks like this



Event on: myCombo

Event: onChange

Function: Custom Code

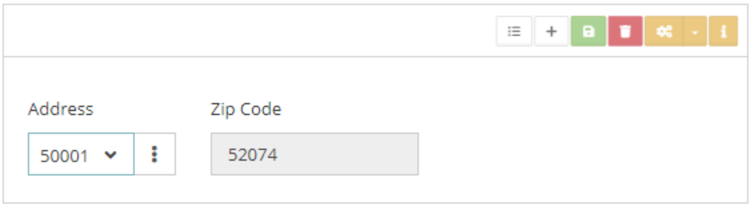
☐ Block input during event execution?

☐ Validate Input before execution?

Code

```
let id = app.getFieldValue("myCombo");
let record = app.loadConfigRecordById("address", id);
app.setFieldValue("nbZipCode", record.data.adrZipCode);
```

In first line of code we are getting the "key field value" of our selected combobox value. To get full record from data we are using "loadConfigRecordById" function in line 2. In line 3 we are setting the value of zip code num box with the value of zip code of loaded record. We also do not want our users to change this value. We need to set "Enabled" styling property of our num box to "No". Now when we select an address from combobox, it sets the num box value with the zip code of record set in combo box.



Address	Zip Code
50001	52074

TabControl

TabControl

When an application has many controls or in other words, a lot of information to store. Adding more controls on one screen, in such manner that they become congested, is not sensible. Instead of doing so, it becomes appropriate to add different pages to your application. Brixxbox provides users with two types of pages: Tabs, and Accordions. In Brixxbox, if we want to add pages to our application we will need a control to manage these pages. This is done by "TabControl".

Documentation

Brixxbox app configurations allows you to add different controls in your app. "TabControl" acts as an host element for TabPages or Accordions. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using this control, "TabControl" control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Size

In Brixxbox, the screen is divided into 12 logical columns horizontally. For each control you can select explicitly how big it should be according to your application needs. Size properties defines how control will be displayed in Brixxbox App.

- Label on Top of Control

- If this option is selected then the label of a control and actual control will be in two different horizontal lines. If this option is not selected then 12 column's will be shared between label and control itself and both of them will be present on same line in side by side formation. For example, label can be 4/12 column's long and control can be 8/12 columns long over same horizontal line.

The display layout is divided into four different categories depending upon the pixels. For each category, Brixxbox allows you to customize the width of control.

- Phone (and up)

- This category includes devices which have screen pixels less than 768 pixels. For example: Smart phones. This will be by default selected category in Brixxbox. In this category, each control will be assigned to 12/12 by default. It means that in extra small screens, each control takes up the whole horizontal line and next control will take up whole next horizontal line and soon.

- Tablets (and up)

- This category includes devices which have screen pixels less than 992 pixels. For example: Tablets. By switching this settings on, they will come into play for particular app.

- Medium Devices (Laptops)

- This category includes devices which have screen pixels less than 1200 pixels. For example: Laptops. In Medium devices, an input control will be set to 4/12 per control by default with option to customize the size of controls.

- Large Devices (Big Monitors)

- This category includes devices which have screen pixels greater than 1200 pixels. For example: Big monitor displays.

- Custom Label class

- This is only relevant if the control is in a From Group, and so the Label is left or right of the control instead of above it. It uses the same rules as the Width properties as above.

Styling

These properties defines how control looks like in Brixxbox.

- Enabled

- It allows two options: yes, no. If "yes" is selected, the control's input will be enabled. In "TabControl" case, it will have no effect because "TabControl" is not expecting any kind of input.

- Disabled in edit mode

- There are two modes available in Brixxbox for each app: edit mode, create mode. In edit mode: user is allowed to add new controls and functionality while in create mode, on the basis of app controls, user is allowed to add a new record, edit an existing record, or list all the records. In order to edit a record, user has to list all the records using app settings and then select a specific record to edit. "Disabled in edit mode" is a checkbox field. If this option is selected then the control will not be functional while editing a previously saved record but it will be functional while creating a new record. In "tabControl" case, it will have no effect because "TabControl" is not expecting any kind of input.

- Visibility

- It will specify the visibility options available in Brixxbox. There are four options available: visible, hidden, hidden in grids, and hidden in forms. For example, if hidden option is selected then we will not be able to different tab pages in a tab control.

- Text Color

- Brixxbox allows user to specify, text colors of controls. There are seven different options available in Brixxbox.

- Tooltip

- Enter a text for a tool tip. This will be translated.

- Help Text

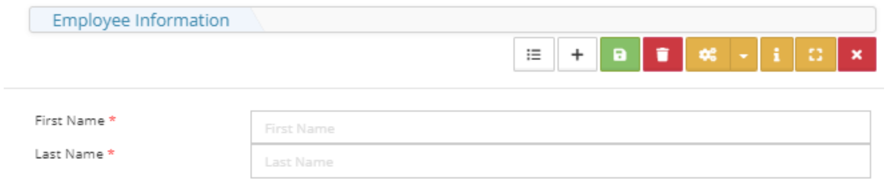
- Brixxbox with the property allow users to add some help text. This text will be displayed below the control. The purpose of this option is to encourage the right behavior of customer while using a specific control.

For further information please have a look at Bootstrap Grid System

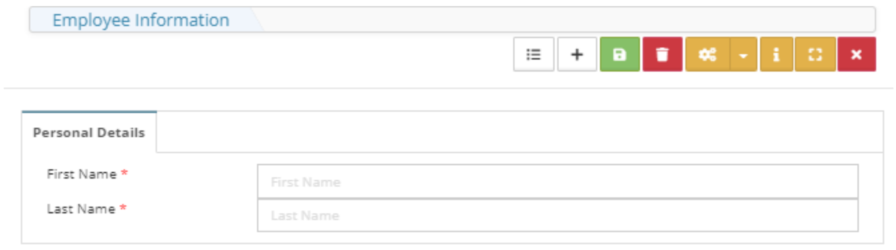
Tutorial

This tutorial assumes that a user has an existing app. In this app, we have two textboxes "first name", "last name". In this tutorial, we want to put them in a tab page and ultimately add this tab page in a "Tab Control". Tab pages are the control containers which can be placed inside "Tab Control". If you want to know more about "Tab Page" have a look at its documentation. Lets get started.

Our app looks like this at start:



Now first add a "Tab Page" control and label it with "Personal Details". In next step, add both textboxes "First Name", "Last Name" as child's to the tab page. Both text boxes have become siblings now. In the last step, add new "Tab Control" and make our tab page a child of "Tab Control"



We have successfully created a tab page and placed it inside the "Tab Control".

TabPage

TabPage

In Brixxbox, a "tab page" is used as a content container. We can use it inside "Tabcontrol". A "TabControl" can contain multiple "TabPages". Each "TabControl" can have any number of controls. One important thing to note here is "TabPage" does not have size properties because in Brixxbox, we use "TabPages" within "TabControl" and they inherit their parent's size properties. The label of each "TabPage" is placed side by side on the top row of "Tabcontrol". By clicking on respective "Tab Control" label, user can access or navigate between different "Tab Pages".

Documentation

Brixxbox app configurations allows you to add different controls in your app. "Tab Page" is used as a container for multiple controls. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using this control, "TabPage" control type should be selected from drop down list. It should be placed as a child to "TabControl". Multiple Tab page's which are placed on same level of inheritance inside a "Tab Control" are called sibling Tab pages.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. For "Tab Page", this property becomes very important. This property is used to recognize content of a tab page. It is recommended to use meaningful labels for a Tab page. For example, if we have an orders app, we can place orders contents in one tab page labeling it as order contents etc.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Styling

These properties defines how control looks like in Brixxbox.

- Enabled

- It allows two options: yes, no. If "yes" is selected, the control's input will be enabled. It has no effect in "Tab Page" because "Tab Page" itself is not expecting any kind of input.

- Disabled in edit mode

- There are two modes available in Brixxbox for each app: edit mode, create mode. In edit mode: user is allowed to add new controls and functionality while in create mode, on the basis of app controls, user is allowed to add a new record, edit an existing record, or list all the records. In order to edit a record, user has to list all the records using app settings and then select a specific record to edit. "Disabled in edit mode" is a checkbox field. If this option is selected then the control will not be functional while editing a previously saved record but it will be functional while creating a new record. In "Tab Page" case, it will have no effect because "Tab Page" is not expecting any kind of input.

- Visibility

- It will specify the visibility options available in Brixxbox. There are four options available: visible, hidden, hidden in grids, and hidden in forms. For example, if hidden option is selected then this specific tab page will be hidden.

- Text Color

- Brixxbox allows user to specify, text colors of controls. There are seven different options available in Brixxbox.

- Tooltip

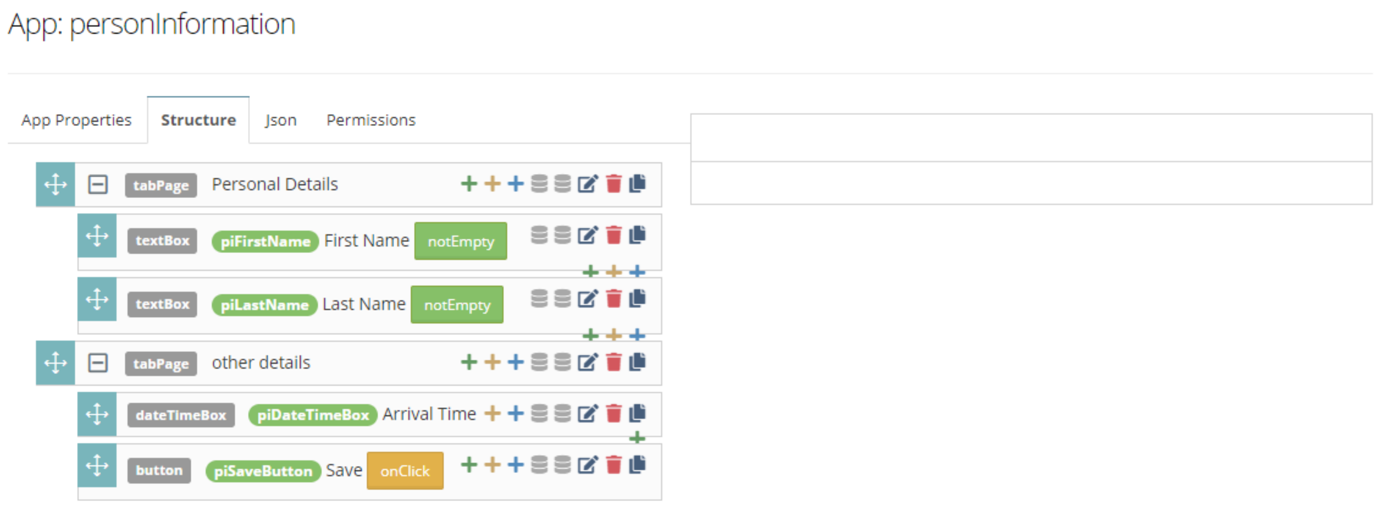
- Enter a text for a tool tip. This will be translated.

- Help Text

- Brixxbox with the property allow users to add some help text. This text will be displayed below the control. The purpose of this option is to encourage the right behavior of customer while using a specific control.

Tutorial

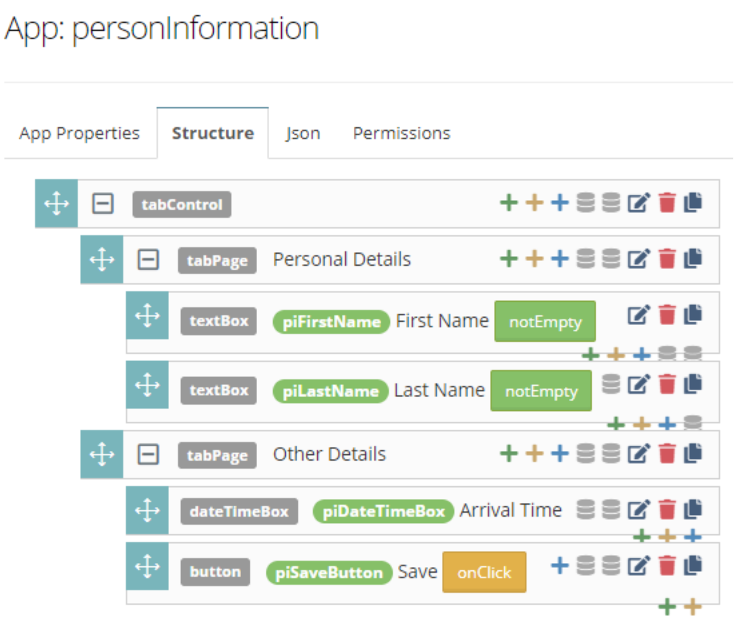
This tutorial assumes that a user has an existing app. In this app, we have two textboxes "first name", "last name", Date Time Box control as "Arrival time", and a "save" button. We use this app to store some information like first name, last name, and arrival date time about our employee. "Save" button validates the inputs and save them, when button is clicked. Now we also want to split personal information (first and last name) and other details (Arrival time) and display them on different pages so that To allow this behavior we need to add two "Tab Pages" to store this information. Add both Tab pages one by one from the drop down list and label them "Personal Details" and "Other Details". Now put "First Name" and "Second Name" text box as a child in "Personal Details" Tab Page. We need to put "Arrival time" Date and Time Box and "Save" button in "Other details" Tab page. Now save the app and lets look at how our application looks now.


App: personInformation

App Properties | Structure | Json | Permissions

tabPage Personal Details
textBox piFirstName First Name notEmpty
textBox piLastName Last Name notEmpty
tabPage other details
dateTimeBox piDateTimeBox Arrival Time
button piSaveButton Save onClick

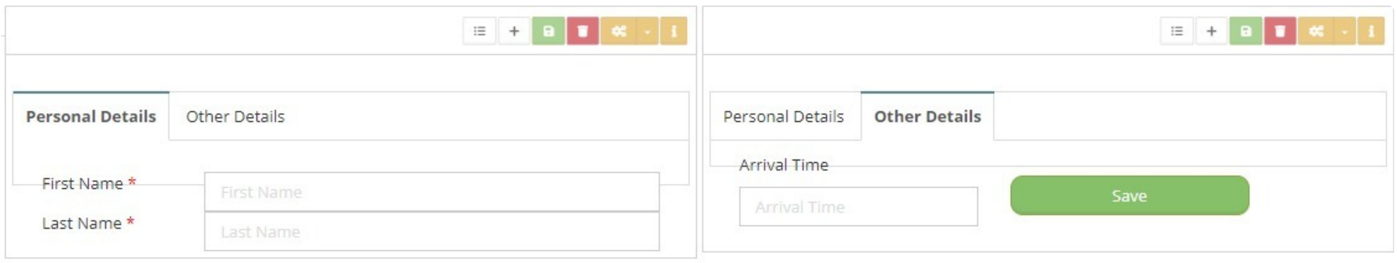
We can see from above figure that we have Tab pages with their respective controls, sound okay but wait a minute! on the right sight of the picture we see nothing. It is showing the our Tab pages are not being shown. The reason is that Tab pages can not be used stand alone. They have to be used inside a "Tab Control" or "Accordion". Lets now add them one by one and analyze changes. Firstly, add a "Tab Control" and makes our two tab pages siblings, inside it. Now save app. App structure will look like this now:


App: personInformation

App Properties | Structure | Json | Permissions

- tabControl
  - tabPage Personal Details
    - textBox piFirstName First Name notEmpty
    - textBox piLastName Last Name notEmpty
  - tabPage Other Details
    - dateTimeBox piDateTimeBox Arrival Time
    - button piSaveButton Save onClick

Now let see how our App looks like:

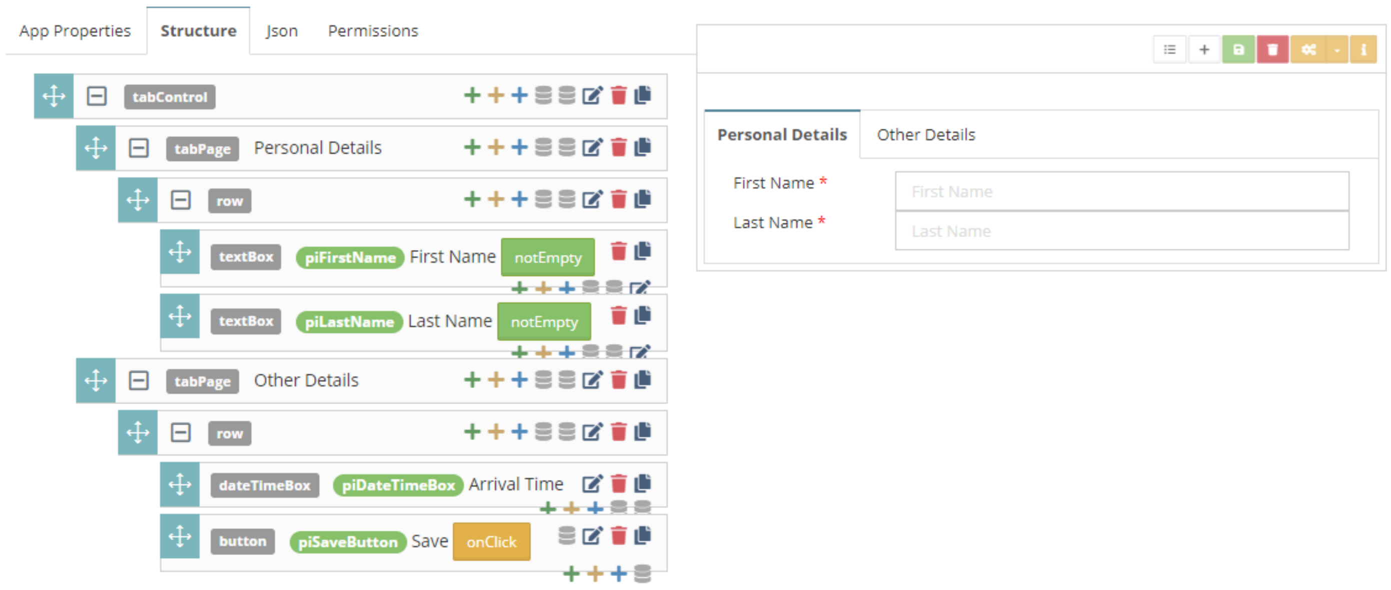

Personal Details | Other Details

First Name * First Name
Last Name * Last Name

Personal Details | Other Details

Arrival Time
Arrival Time
Save

On the left side, we have selected "Personal Details" Tap page while on right side, we have "Other Details" tab page. We can only select one tab page at a time. We are almost done. As you see in above picture, there is a horizontal line passing through "First Name" control on left side and from "Arrival Time" on right side. It does not look beautiful. Now let get rid of it. To overcome this we have to use "Row" control. Lets add a row control and analyze App structure and App view.


App Properties | Structure | Json | Permissions

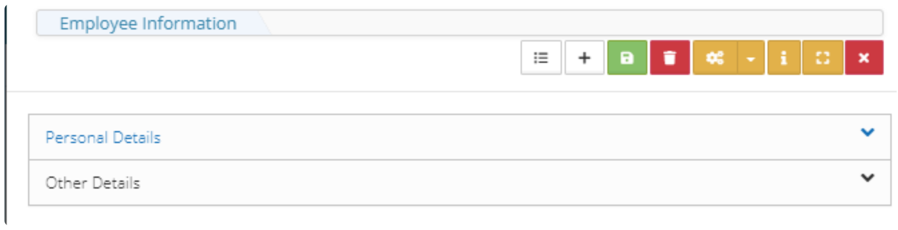
- tabControl
  - tabPage Personal Details
    - row
      - textBox piFirstName First Name notEmpty
      - textBox piLastName Last Name notEmpty
  - tabPage Other Details
    - row
      - dateTimeBox piDateTimeBox Arrival Time
      - button piSaveButton Save onClick

Personal Details | Other Details

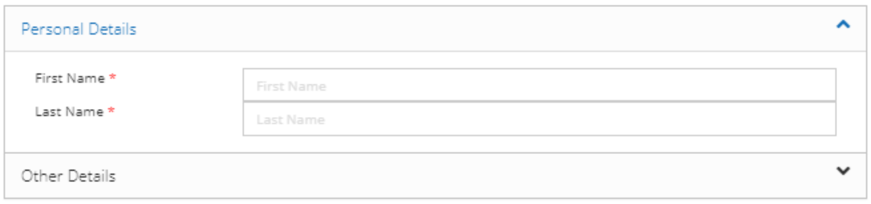
First Name * First Name
Last Name * Last Name

On the left side, we can see that we added "Row" control as a child of Tab pages and this row acts as a parent of other controls. On the right side we see that there is no line our controls are present in a container. Beautiful!

Let's see, how our Tab pages will look like when placed inside "Accordion" instead of "Tab Control". Add an "Accordion" in place of "Tab Control" in our app and everything else remains the same:



From above figure, we can see our tab pages but only their names are visible. What happened here? Ahh, remember that "Accordion" control makes our tab pages expandable and compressible. By default they are in compressed form. Now lets expand "Personal Details"



Now we can see the contents of Personal Details Tab Page. If you want to know about "Tab Control" or "Accordion", please have a look at their documentation also.

TextBox

Text Box

A text box just lets the user enter a text and saves it. You can use various validators to check the input. For example, "nonEmpty" validator will display error that this field should not be left empty.

Documentation

Brixxbox app configurations allows you to add different controls in your app. Text box is one of the simplest and commonly used control. Its general properties are described below

General Properties

These properties defines how control behaves in Brixxbox.

- Control Type

- For using Text Box control, "text input" control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Data

- This property specifies from where this control gets data. To set it, specify the controlId of data source. By default "Not in Database" option is selected.

- Include in global search

- If checked, this allows this control to be included in global search scope of this application.

- Select all on focus

- This property when checked allows to select all the text in the text box, whenever this text box comes in focus.

- Input Mask RegEx

- This property allows you to set regular expression for input of text box. For example, **^[0-9a-zA-Z]*$**. This regex allows digits from 0 to 9, small and big alphabets to be part of the input.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Unit

- The property allows you to specify a unit character here. For example we can specify $ for US Dollar, € for Euro etc.

- Load record on load

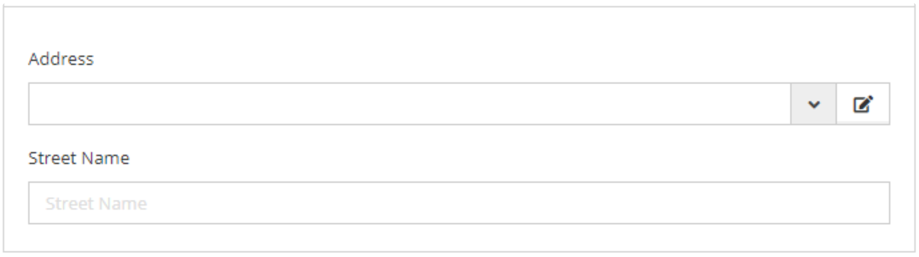
- If this checkbox is selected then Brixxbox will load the record using value of this text box as a key.

- Default value

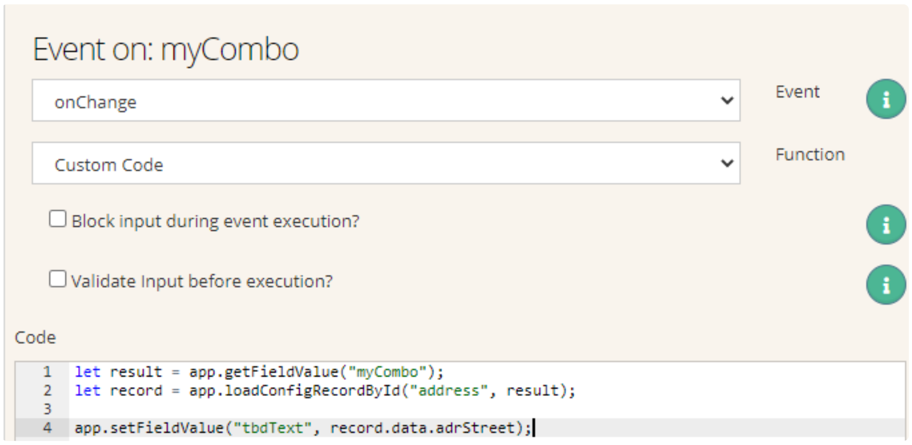
- This value is set when an application is opened or a new record is created.

Tutorial

This tutorial assumes that a user has an existing app and they want to add a new control of type "Text Input". First of all, select "Text Input" from list of controls. After that you need to give a id to your textbox and create column in database if you want to store the input in database. Here we named it "tbdText". Also, add a public label for Textbox. I labelled it "Street Name". You can any text in it. For example, add name of street. When you save this record. It will be saved in database. This is the first usage. We can also use text box to display the results from other controls. Lets use address combobox here. It is the same combobox we used in combobox tutorial.



Now when we select the value of combobox, we also want to set the value of "tbdText". The value of "tbdText" will correspond to name of street of the address selected in combobox. For this purpose, we need to add "onchange" event on combobox and add a custom code for it. In this code, first of all, we get the id of address chosen in combobox then we load the complete record using Brixxbox "loadConfigRecordById". In the end we will set the value of "tbdText" by using "setFieldValue" function. In this function, the first parameter takes the controlId of control whose value we want to set and second parameter takes actual value.


Event on: myCombo

Event: onChange

Function: Custom Code

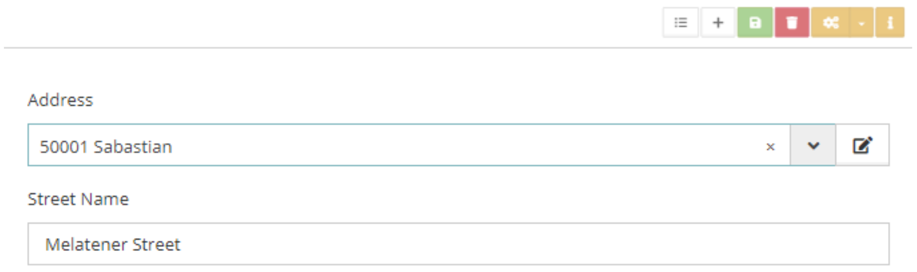
☐ Block input during event execution?

☐ Validate Input before execution?

Code

```
let result = app.getFieldValue("myCombo");
let record = app.loadConfigRecordById("address", result);

app.setFieldValue("tbdText", record.data.adrStreet);
```

Now save the app config and test it by selecting any value of address combobox. It will also set the street name text box with corresponding street name.


Address: 50001 Sabastian

Street Name: Melatener Street

LinkLabel

Just like a label but it will create a clickable link.

RadioButton

The RadioButton control is one element of a radio button group.

Group Name

- The Group Name binds different radio buttons controls. All radio buttons with the same group name will interact. E.G. switching one on will switch others of the same group off.

Unique Value

- Each radio button within a group must have a unique value.

Default setting for button group

- One button within a group can be marked as the default value. This will set the button initially to on.

Example

We could define radio buttons for colors. The group name could be "color" and we can set up different buttons with unique values like "red", "blue", "green", ...

Report

This control will preview a report.

Row

Most of the controls are linked with storing and displaying information but row is purely structural element. It is used to group fields.

Fields in same row will be then placed next to each other where ever it is possible. This grouping is in horizontal format as the name of the control suggests itself. If there is no space in the same row, next element will be placed in the next adjacent row. There is no visible boundary of row control.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Row" control type from drop down list. As this most commonly used control, Brixxbox has a designated button for adding a new row. User can find this button on app editor page and on the top right corner.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will see how to use row control. this control is purely structural. For this tutorial, we will be using "unitDemo" app. We have used this app in "unit" control tutorial also. If you want to look into details of this app please go through the "unit" documentation. Here is the link: | Unit Documentation. Our app looks like this:

Now we want to separate item id and item name from the other properties. This is where row comes in to place. We need to select row control from controls list and put both item id and item name as child's in this row:

Now you can see the controls in the app are rearranged. In first row, only item id and item name controls are present. All other controls are separated. Now lets say we also want to add sale price and sale price test control's to one row also. For this, select row control form "row button". It is present on top right corner, just click it and it will add row. Now put sale price controls in this row:

Now you can see the price controls are rearranged in the app.

Scanner

This control is an element where you can use the phone camera to scan a barcode.

This control triggers the onScan event when it scanned a barcode.

SignaturePad

With a signature pad, you can add signatures to your app

Use a longtext column to store the signature.

Tag

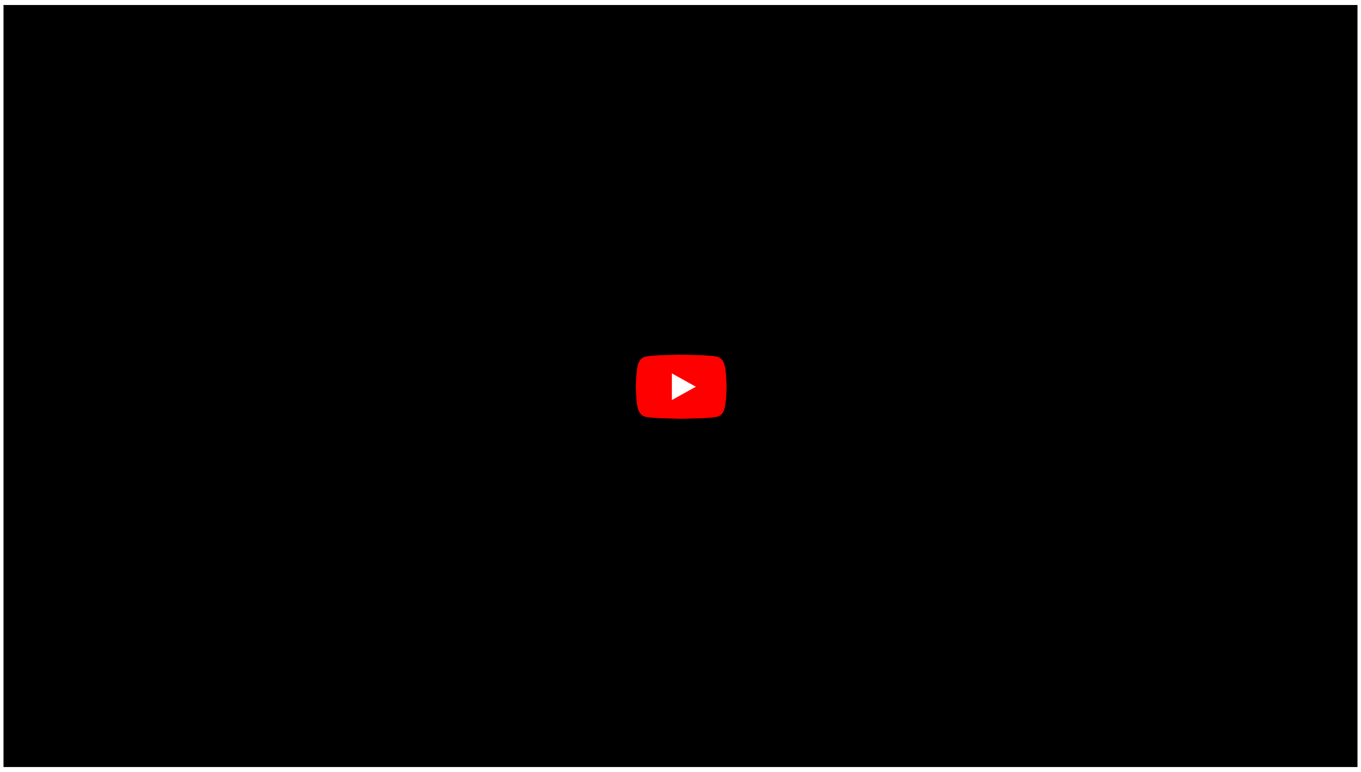
This is a tag control where the user can choose from a list of tags or add new ones.

TemplateGrid

A template grid is a list of records. Each record will be displayed based on the child templateGridElement Controls.

You can add css to this control, that works with all the templateGridElements.

Demo

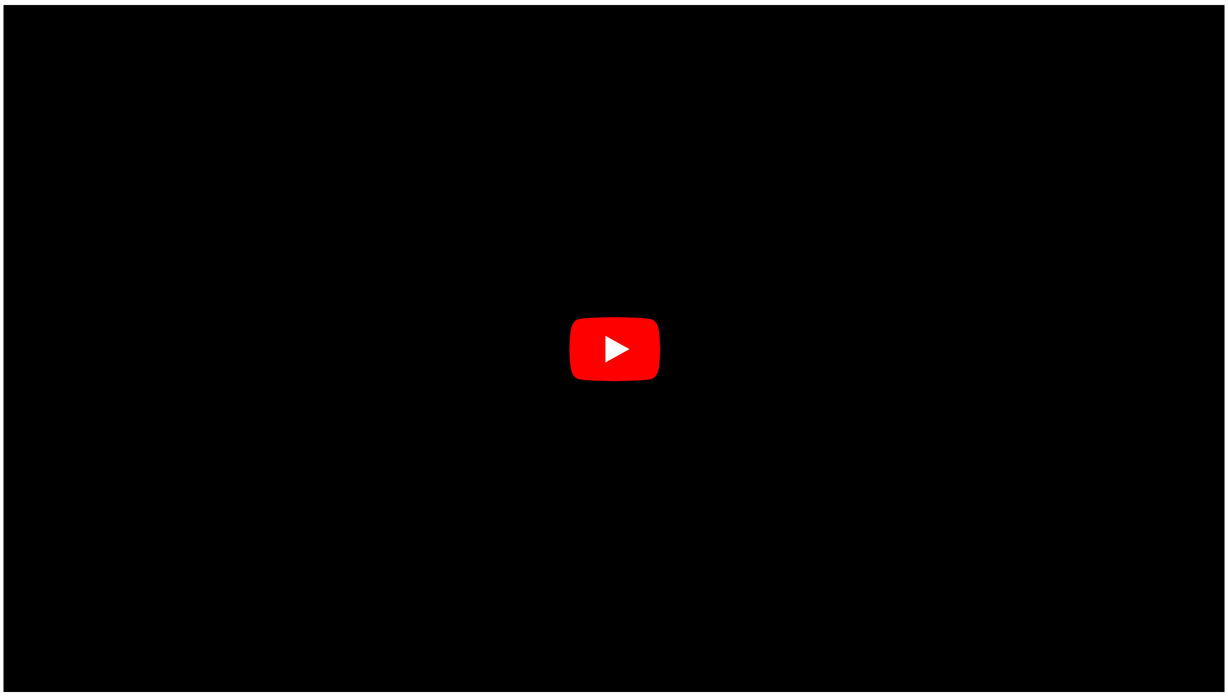


TemplateGridElement

Placed as a child unter templateGrid, provides templateGridElement the html code to render the record

If there are more than 1 templateGridElement und a templateGrid. The user can choose the displayed version.

Demo



TimeBox

Allows user to enter a time of day (not a timespan).

Unit

The unit control allows you to show a unit from a data source

Brixxbox allows you to have a unit property for each numeric box control. It is a character based property. User can set any character and it will be displayed as a unit with this numeric control. The is one way of assigning unit to a control. Brixxbox also provides it users with a unit control. The only purpose of this control is to load unit from a data source In this way, Brixxbox allow users to store numeric quantities with their units also. For example, this control can be placed as a child control to a numbox control.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Unit" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will be using "unitDemo" app. This is very simple app. It's purpose is to elaborate different ways to set unit of different controls. In this app, we have two controls and both of them are numeric. The first field is Item Id, it will be used to store id's of items and second field will be used to store the price of corresponding item. Our app looks like this

As you can see our "Sales Price" is numeric but we do not have any unit with it. Brixxbox provides us with unit property with some controls like "numbox". In this property we can assign any character to make it a unit of specific control. Now, we need to open "Sales Price" properties and update unit to "Euro" and then save the app. Now we can see in the app editor that on left side of "Sale Price" numbox Euro symbol is being displayed.

This is a nice way to assign unit to a specific control but this is control specific and we do not store in our database. Lets assume we want a unit app so that we list all the units available in our database. This can be done in two steps. First, we need to add an "unit" app which stores the names of all the units in it.

It simply stores the names of units we want in our app. Lets quickly add three units in it: Pc for piece, $ for Us Dollar, and € for Euro currency. In second step, we need to add a combobox named "Unit Type" and assign our unit app as its "sub data source" with id, and unit name as combo box key and field values.

Now as you can see from above snapshot, our unitDemo app has a combo box and now we are also seeing all available units which are present in our unit app because we have made unit app, our combobox sub data source. This is another way to display unit next to numeric field. We have already discussed combobox in our tutorial for combobox. If you you need any help for combobox, please visit this combobox documentation link ComboBox Control Documentation.

Now we proceed to our third way of presenting unit with numeric control. Here "unit" control comes into play. Let suppose we want display the price of item in another control. This can be a different app also but for the sake of simplicity we will add new numeric control in same app. We also make new control, lets call it "sale price test", read only. We want a simple functionality that as soon as the value of "sale price" changes, it will reflect on "sale price test" control also. We can add onChange event on "sale price". This event on value change will get "sale price" value and set it as the value of "sale price test".

Now we also want a unit for this control. For this we need to a unit control in our app and make it a child control of our "sale price test" control. We also need to assign a data source to our unit control. Lets add our "unit app" as a data source. This data source will allow it to query and get a specific data from data source.Our app looks like this

Now we can see that our "Sale Price Test" has a box for unit but it is empty. The reason behind this is that we need to add query to get specific unit from data source. Lets assume we want same unit as selected in our unit type combobox. The query is

Now when we change the value of unit in our unit type combobox, this change also gets reflected in our sale price test unit.

WedgeScanner

Reads scanns from a wedge barcode scanner and will trigger the OnScan event.

The minimum length for a scan result (including prefix and suffix chars) is 5 characters. That means: A scanner with prefix and suffix chars of '#' can scan a barcode, that is at least 3 characters long. A scanner without a prefix and suffix char will scan a barcode that is 5 or more characters long. In case you need to scan a shorter barcode, you need to add prefix or suffix characters in your scanner settings

Widget

A Widget is another structural control in Brixxbox controls. It is used to group controls.

It is a structural control like groupbox. For a vertical alignment, you can use formgroup, groupbox. Widget is the advanced form of formgroup. User can use it inside a WidgetContainer also. Like groupBox, it also has a header and a border.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Widget" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will see what a widget is and how to use it to achieve vertical grouping of controls. This control is purely used for structuring controls. For this tutorial, we will be using item app. Our app has 2 controls: item id(numbox), | item name(text box). Our app looks like this:

Now lets see what an empty widget look like. Add a widget control from controls list and add label "Item details" to it:

It is exactly similar to groupbox. In empty condition it has a label and a visible boundary. Now we want to group item id and item name. For that, we need add both item id and item name inside widget. Now these two controls with be placed inside it and if there are other controls present, Brixxbox will try to place other controls in horizontal fashion, wherever it finds a place. Our app now looks like this:

As we can see in the above snapshot that item id and item name controls are placed inside widget and widget containing them has a clear boundary and a header.

WidgetContainer

A Widget container is container that can host multiple widgets just like tabcontrol hosts multiple tabs in Brixxbox controls. It is used as a parent to widgets. It is a structural control. User can use it to place and group widgets inside it also.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Widget Container" control type from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control. It should be meaningful.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

Tutorial

In this tutorial, we will see how widget container can group different containers. This control is purely used for structuring controls. For this tutorial, we will be using item app. Our app has 4 controls: item id(numbox), | item name(text box), , sale price(numbox), and quantity(numbox). Our app looks like this:

Now we need to add to widgets. Add widget controls from controls list and add label "Item Ids", and "Item Details" to them. Our app looks like this now:

In above snapshot we can see that two widgets item ids and item details are placed side by side in a horizontal fashion and also the controls inside widgets themselves are in horizontally distributed. Widget container can help us organizing our controls in better way by adding extra hierarchy level to them.

WysiwygText

This is a control, that lets you enter html formatted text

You can store that typically in a longtext column.Wysiwyg stands for „**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et“.

Events

onAppStart

The event occurs when an app is started but before a record is loaded and the OnAppInitialized event is fired. Default Values are set at this point.

```
console.log("app is started and initialized");
```

onAppInitialized

The event occurs when an app is initialized. The OnAppStarted event is processed at this point. If a record id was given as a start parameter, the record is loaded and displayed at this time.

```
    console.log("app is initlialized");
```

onAttachmentDeleted

The event occurs when a user deletes an attachment from the right sidebar. The delete attachment function does not trigger the event.

```
console.log("An attachment was deleted");
```

onAttachmentsShow

This event is called just before the attachments panel open. Can be used to set the config and id of the attachments you want to show. This is an App wide event!

Example Usages

```
    app.attachmentRecordId = app.getFieldValue("addressId");
    app.attachmentAppName = "address";
```

onAttachmentsHide

The event occurs when the right sidebar for attachments gets closed

```
    console.log("An attachment sidebar closed");
```

onChange

The event occurs when a control content is changed. You can use app.isLoadingRecord if you need to know if a record is loading and that is the reason for the change event.

```
console.log("Change detected");
if(app.isLoadingRecord){
   console.log("The app is displaying a record and that triggered the event");
}
```

onCellClick

The event occurs when a grid cell is clicked. You can get more information where exactly the user has clicked.

```
    let rowElement = app.getFieldValue("myGrid.clickedRow"); //The html Row element (a tr)
    let cellElement = app.getFieldValue("myGrid.clickedCell"); //The html Cell element (a td)
    let cellId = app.getFieldValue("myGrid.clickedCellId"); //column id of the clicked cell (e.g. "adrFirstName")
```

onClick

The event occurs when a control is clicked.

```
    console.log("Hello:" + app.getFieldValue("adrName"));
```

onChildAppClosed

This event is called when a child app of you current app is closed. The closed app object is available under **eventArgs.details** in the even source code- But be aware, that this app is closed. You can access properties liek appName but you might not use saveConfigRecord or such methods on the closed app.

Example Usages

```
    app.refreshDatasource("orderQuantity");

    console.log("The app: ${eventArgs.details.appName} was closed");
```

onDataTransform

The event can be used to transform a datasource line.

This function is often used to transform a calender event object. Called once pre received record. Must return a calender event.

Example 1

Modifiy an existing record, to keep all the information but add new properties for the event

```
eventArgs.details.title = eventArgs.details.terTitle;
eventArgs.details.description = eventArgs.details.terLongText; //This will be the tooltip text
eventArgs.details.start = eventArgs.details.terStart;
eventArgs.details.end = eventArgs.details.terEnd;
return eventArgs.details;
```

Example 2

Create a new record.

```
return {
    start: eventArgs.details.terStart
    end: eventArgs.details.terEnd
    title: eventArgs.details.terTitle
}
```

onEventClick

The event occurs the user clicks on one of the events. getFieldValue('controlId.changedEvent') or eventArgs.details ca
determin the clicked event

```
app.startBrixxbox({
      appName: "appointmentDetails",
      id: eventArgs.details.id, //eventArgs.details provides the clicked event object
      startMode: "modal"
   });
```

Google Calendar

Here are some usecases for event clicks you might find helpful. In Case you are using a google calendar, you could open the calendar event in a nev

```
      window.open(eventArgs.details.url, '_blank', 'width=700,height=600'); //this is our custom operation. we will open the event
```

Or you can return false at the end of the event source code, which means, that you did not handle the event by yourself and there for to execute the de

onEventChange

The event occurs the user moves or modifies an event. getFieldValue('controlId.changedEvent') or eventArgs.details can be used to determin the changed event

You can get mor information about the effects:

1. the event before the change: getFieldValue('myCalendarControl.oldEvent')
2. other events that got affected: getFieldValue('myCalendarControl.relatedEvents')

You can call cancelCalendarChanges to cancel all the changes

```
let eventObj = app.getFieldValue("myCal.changedEvent");
let record = app.loadConfigRecordById("appointment", eventObj.id);
record.data.aStart = eventObj.start;
record.data.aEnd = eventObj.end;
app.saveConfigRecord("appointment", record);
```

onFileImport

The event occurs when an file was selected on a fileImport control. The import function will detect if the file is using a comma or a semicolon as separator. Brixxbox will therefore count the occurrences of commas and semicolons in the first line (Header) and will use the higher counted char as separator.

You can access eventArgs.details inside this event to get the file content and meta data:

- eventArgs.details.csvLines - Array of all lines splitted by csv rules
- eventArgs.details.lines - Array of all lines splitted by csv rules
- eventArgs.details.fileExtension - extension of the import file ("csv" for example)
- eventArgs.details.fileExtensionLower - extension of the import file in lowercase
- eventArgs.details.fileInfo - a javascript file info object
- eventArgs.details.fileName - the file name
- eventArgs.details.fileNameLower - the file name in lowercase
- eventArgs.details.fileNameFull - the file name with extension
- eventArgs.details.fileNameFullLower - the file name with extension in lowercase
- eventArgs.details.rawData - the whole file content as a single text
- eventArgs.details.rawLines - an array of all the lines. Use this if you want to split the lines following your own rules

```
for(let i = 0; i < eventArgs.details.lines.length; i++){
    console.log(`This is line ${i} with name: ${eventArgs.details.lines[i].myNameField}`);
}
```

onKeyDown

Further information about key events

```
   console.log(event);
/*
event.type
event.target
event.which
event.altKey
event.keyCode
event.shiftKey
event.ctrlKey
*/
```

onKeyPress

The event occurs when a key is pressed. You can access the event information, using the event parameter.

Further information about key events

```
   console.log(event);
/*
event.type
event.target
event.which
event.altKey
event.keyCode
event.shiftKey
event.ctrlKey
*/
```

onModalClose

The event occurs when an app closes from a modal state.

```
    console.log("app closed")
```

onMailHistoryShow

This event is called just before the mail history panel open.

Example Usages

```
    console.log("mail history panel opened");
```

onRowClick

The event occurs when a grid row is clicked. You can get more information where exactly the user has clicked.

```
    let rowElement = app.getFieldValue("myGrid.clickedRow"); //The html Row element (a tr)
    let cellElement = app.getFieldValue("myGrid.clickedCell"); //The html Cell element (a td)
    let cellId = app.getFieldValue("myGrid.clickedCellId"); //column id of the clicked cell (e.g. "adrFirstName")
```

onRecordLoad

The event occurs right before a record is loaded. By returning false, the following load operation is canceled

```
console.log("loading a record");
if(a != b){
    return false; //cancel the load
}
```

onRecordLoaded

The event occurs after a record is loaded and displayed. You could do some updates or refreshes at this point.

```
    console.log(`record ${app.recordId} successfully loaded`);
    console.log(app.record); //you can access the record json object here
```

onRecordSave

The event occurs right before a record is saved. By returning false, the following save operation is canceled

```
console.log("saving a record");
if(a != b){
    return false; //cancel the save operation
}
```

onRecordSaved

The event occurs when a record is saved. You might want to do some additional tasks in that case.

```
console.log("Record saved");
```

onRowSelectionChanged

Scope: grid The event is triggered when the selection of grid lines change. When a line is selected or deselected.

```
//Enable a button if exactly 1 line is selected
let selectedRows = app.getFieldValue("myGrid.selectedRows");
app.setEnable("myButton", selectedRows.length === 1);
```

onRowCreated

The event occurs when a grid control displays a row. RowCreated does not mean that a database record is created.

The are a couple of special objects you coul use in this event type:

- eventArgs.details.row - this is the new row as a html element.
- eventArgs.details.data- this is the record used for this row
- eventArgs.details.index- this is the index of the row in the entire grid

```
    if(eventArgs.details.data.cordlnShippedQuantity >= eventArgs.details.data.cordlnOrderQuantity){
        $(eventArgs.details.row).addClass("success");
    }

    if(eventArgs.details.data.cordlnOrderQuantity - eventArgs.details.data.cordlnShippedQuantity > eventArgs.details.data.cordlnSt
        $(eventArgs.details.row).addClass("warning");
    }
```

onRecordNew

The event occurs on app level as well as on a grid control.

App Level

On app level, this event is mostly used to initialize certain fields.

Grid Control

eventArgs.details gives you the startBrixxbox parameters befor the brixxbox for the new record is started. You could add additional values to the start object or change the app to use.

```
eventArgs.details.additionalValues.sesTrackId = app.getFieldValue("eveTrack");
eventArgs.details.additionalValues.sesTrackConfigId = app.getFieldValue("eveTrackConfig");
```

onRecordDelete

The event occurs right before a record is deleted. By returning false, the following delete operation is canceled

```
console.log("deleting a record");
if(a != b){
    return false; //cancel the delete
}
```

onRecordDeleted

Scope: grid, app wide The event occurs when a record is deleted. The ids of the deleted records are passed as an array in eventArgs.details for example: ["1", "5", "10007"]

```
console.log('Deleted records: ' + eventArgs.details);
```

Another example to iterate over the deleted record ids:

```
for(let i = 0; i < eventArgs.details.length; i++){
    console.log("Deleted id: " + eventArgs.details[i]);
}
```

onKeyUp

The event occurs when a key is released. You can access the event information, using the event parameter.

Further information about key events

```
   console.log(event);
/*
event.type
event.target
event.which
event.altKey
event.keyCode
event.shiftKey
event.ctrlKey
*/
```

onScan

The scan events gets triggered when a barcode (or 2D code) is scanned, e.g. by using the wedgeScanner control. You can then ask the control for the scanned code.

Simple Example

```
    console.log("The scanned code is: " app.getFieldValue("myScannerControlId"));
```

Example using the GS1 Splitter business function

```
let result = await brixxApi.businessBrixx({
     functionName:"Gs1Splitter",
     methodName: "Split",
     gs1Code: brixxApi.getFieldValue("myScannerControlId")
 });
 console.log(result)
```

onReturnFromModal

The event occurs when a modal child app is closed. This is a good opportunity to refresh some data because the child app might have modified it. Remember: Grids are refreshed automatically if you return from a modal app. The closed app object is available under **eventArgs.details** in the even source code- But be aware, that this app is closed. You can access properties liek appName but you might not use saveConfigRecord or such methods on the closed app.

```
    app.refreshDatasource("orderQuantity");

    console.log("The app: ${eventArgs.details.appName} was closed");
```

onSubDataRequest

This event is called when a subDataSource is requested. You can return the data here instead of fetching it from the server automatically

Example Usages

```
return [
    {
        id: 1,
        name: "Germany"
    },
    {
        id: 2,
        name: "Netherlands"
    },
    {
        id: 3,
        name: "Poland"
    },
]
```

Example Usages 2

```
return [
    (app.loadConfigRecordById("item", 1)).data,
    (app.loadConfigRecordById("item", 2)).data,
    (app.loadConfigRecordById("item", 3)).data,
]
```

Example Usages 3

```
return [
    {
        id:1,
        itmName: "lorem",
        itmPrice: 1.23,
        itmReleaseDate: moment().add(1, "days").toISOString(),
        itmDescription: "some Text",
    },
    {
        id:2,
        itmName: "ipsum",
        itmPrice: 4.56,
        itmReleaseDate: moment().add(5, "days").toISOString(),
        itmDescription: "some more Text",
    }
];
```

onTabShown

The event occurs when a tab page comes to the visible state. You might want to refresh some data here.

```
    console.log("The tab is visible");
```

onTimeSelected

The event is triggered when a empty time period is selected by the user. You can use this to create a new calendar event

```
app.startBrixxbox({
      appName: "termin",
      startMode: "modal",
      additionalValues: {
         terStart: eventArgs.details.startStr,
         terEnd: eventArgs.details.endStr,
      },
   });
```

onPictureTaken

Event that gets fired, when a camera control has taken a new picture

Example usage for onPictureTaken

Print picture

```
 brixxApi.printBlob(brixxApi.getFieldValue("controlId"));
```

Add picture to image control

```
brixxApi.setFieldValue("controlId image control", brixxApi.getFieldValue("controlId camera control"));
```

Add picture to attachements

```
brixxApi.uploadAttachement(brixxApi.getFieldValue("controlId"), documentTypeId(opt
```

OnCheckPermissions

OnCheckPermissions

The event occurs once after OnAppInitialized, when a possible record, that came as a parameter is loaded. You can put your code, where you check all your permission changes here, and trigger this event in OnChange events manually if you need to.

```
    console.log("app is started, initialized, and a record may be loaded");
```

Tips and Tricks

Date and Time

The moment.js Library

brixxbox includes the following client side libraries to deal with date, time and durations and calculations in that area.

- moment.js

```
//format a date
var myMoment = moment(app.getFieldValue("myDateField"));
console.log(myMoment.format("lll")); //Jul 13, 2020 1:02 PM
```

- moment duration format

```
//simple format a duration
moment.duration(123, "minutes").format(); //2:03:00
```

```
//how old is the value in myDateField
var myDuration = moment().diff(moment(app.getFieldValue("myDateField")), "seconds");
moment.duration(myDuration, "seconds").format();
```

Server API Reference

How to Use Server Side Functions

Preparing to use BrixxServerApi

Install from npm. This will give your project access to the BrixxServerApi.

```
npm i @brixxbox/brixx-server-api
```

Reference the Class BrixxServerApi from your function source.

```
const BrixxServerApi = require('@brixxbox/brixx-server-api');
```

Create an instance from BrixxServerApi to access the avialable methods.

```
//Note: the constructor will always get the 2. parameter from
//the functions call as parameter. This parameter expects the brixxboxMeta
//as part of the body content (req.body.brixxboxMeta)
const brixxServerApi = new BrixxServerApi(req);
```

Now the different methods from brixxServerApi can be called.

Example

```
const BrixxServerApi = require('@brixxbox/brixx-server-api');

module.exports = async function (context, req) {
    const brixxServerApi = new BrixxServerApi(req);
    currencyRate = await brixxServerApi.businessBrixx({
        functionName:"CurrencyConverter",
        methodName: "Convert",
        fromCurrencySymbol: "EUR",
        toCurrencySymbol: "USD"
    });

    //result back to caller (if needed)
    context.res = {
        status: 200,
        body: {
            responseMessage: 'Rate EUR/USD: ' + currencyRate,
            currencyRate: currencyRate
        }
    };
}
```

Server Side Api Function List

- loadConfigRecordById
- saveConfigRecordById
- sqlWrite
- sqlRead
- sendEmail
- createReportPdfBlob
- cloudPrint
- businesssBrixx

To invoke a server function from the a brixxApi client please check serverFunction

Functions

loadConfigRecordById

Server Function to load a config record and return the record object.

Parameters

1. mode - Config name
2. id - Record Id

Example Usages

loads a record and stores it in a variable

```
    let record = await brixxServerApi.loadConfigRecordById("address", req.body.parameters.addressId);
```

See also client side api loadConfigRecordById

saveConfigRecordById

Saves or creates a config record and returns the new record object.

Parameters

1. mode - Config name
2. id - Record Id

Example Usages

Create a new Record with 2 fields

```
    let newRecord = await brixxServerApi.saveConfigRecordById({
        adrName: "John",
        adrLastName: "Doe",
    });
```

See also client side api saveConfigRecordById

sqlWrite

Executes a write sql statement and returns the number of rows affected

Parameters

1. statementName - The name of the statement
2. additionalParameters - all controls in the current config are set as paramters automatically. If you need to add additional parameters, you can use this json object to set them
3. queryOptions - (optional) a json object with options for the request
   - timeout - (optional) timeout for the SQL Request. Default is 30 seconds
   - connectionKey - (optional) a key to a custom settings entry with the connection string to an external MSSQL database

Example Usages

Update a record with parameters

```
    let rowsChanged = await brixxServerApi.sqlWrite("updateAddres", {
        newName: "John",
        addressId: 45,
    });
```

See also client side api sqlWrite

sqlRead

Executes a write sql statement and returns the number of rows affected

Parameters

1. statementName - The name of the statement
2. additionalParameters - all controls in the current config are set as paramters automatically. If you need to add additional parameters, you can use this json object to set them
3. queryOptions - (optional) a json object with options for the request
    - timeout - (optional) timeout for the SQL request. Default is 30 seconds
    - connectionKey - (optional) a key to a custom settings entry with the connection string to an external MSSQL database

Example Usages

Read all data without parameters

```
let rowsChanged = await brixxServerApi.sqlRead("getAddresses", {}, {timeout: 90});
```

Read all data with parameters

```
let rowsChanged = await brixxServerApi.sqlRead("getAddresses", { adrType: 1});
```

See also client side api sqlRead

sendEmail

This server side api function does support the same parameters (Attachements and so on) as ComposeEmail from the client side api. Emails are sent automatically, as there is no UI.

Example 1

```
    await brixxServerApi.sendEmail({
            to: ["john.doe@acme.com"],
            text: "Hello World",
            subject: "Demo Email",
        });
```

See also client side api composeEmail

createReportPdfBlob

This server side api function does support the same parameters as createReportPdfBlob from the client side api.

Example 1

```
    await brixxServerApi.createReportPdfBlob({
            reportName: "myDemoReport",
            configName: "address",
            archive: true,
            parameters:{
                id: 45
            },
        });
```

See also client side api createReport

cloudPrint

This server side api function does support the same parameters as cloudPrint from the client side api.

Api Version > 0.1.30

Example 1

Send a blob object to the cloud print api

```
    let resultObj = await brixxServerApi.cloudPrint({
            blob: myBlob,
            printerName: "HP LaserJet",
        });
    console.log(resultObj.status);
```

Example 2

We create a blob with create report and send it to the cloudPrint api

```
        let pdfBlob = await brixxApi.createReportPdfBlob({
            reportName: "demo1",
            configName: "address",
            archive: true,
            parameters:{
                id: 45
            },
        });
        let resultObj  = await brixxApi.cloudPrint({
            blob: pdfBlob,
            printerName: "HP LaserJet",
        });
        console.log(resultObj.status);
```

Example 3

Send a base64 file content to the cloudPrint api.

```
        let resultObj = await brixxApi.cloudPrint({
            documentBase64Content: myBase64PdfContent,
            printerName: "HP LaserJet",
        });
        console.log(resultObj.status);
```

See also client side api cloudPrint

businessBrixx

This server side api function does support the same parameters as businessBrixx from the client side api.

```
let result = await brixxApi.businessBrixx({
            functionName:"CurrencyConverter",
            methodName: "Convert",
            fromCurrencySymbol: "EUR",
            toCurrencySymbol: "USD"
        });
console.log(result);
```

See also client side api businessBrixx

Configuration

Workspace Settings

CustomHeaderFooterBackgroundColor

CustomHeaderFooterBackgroundColor

Set the [css color] for the header and footer bar

Example values:

1.
    - rgb(255, 123, 0)
    - #ff12ab
    - lightBlue

If setting a color is not enough any valid css style can be applied.

CustomInviteText

CustomInviteText

You can set the text body for the invitation emails to introduce them to your workspace.

CustomLogoBackgroundColor

CustomLogoBackgroundColor

Set the [css color] for the logo background in the upper left corner

Example values:

1.
   - rgb(255, 123, 0)
   - #ff12ab
   - lightBlue

If setting a color is not enough any valid css style can be applied.

CustomLogoUrl

CustomLogoUrl

Enter a url to a log file, to display in the upper left corner of the brixxbox

The image file should be similar to [the brixxbox logo image]

RecordLockWaitTime

RecordLockWaitTime

Defines the time in seconds, a user has to react on a record edit request from another user. The default is 120 seconds, that means if a editing user is asked to release the record, the user has 120 seconds to accept or decline the request. after 120 seconds the request will be forcefully accepted.

WhiteLabeling

White Labeling

In this Workspace settings section, you can configure the logo image, header- and footer color to brand your workspace if you have a partner subscription

App Editor

Custom HTML Templates

jsTree

Tree Control

```
{
  "id": "costCenterTree",
  "controlType": "htmlTemplate",
  "refersToConfigSource": "",
  "htmlTemplateContent": "<style>\r\n    .bbjsTree{\r\n        \r\n    }\r\n</style>\r\n<input type=\"text\" value=\"\" class=\"
  "events": [
    {
      "name": "onSubDataRequest",
      "code": "debugger;\r\n//Lib laden, wenn noch nicht passiert\r\nif (!window.jsTree) {\r\n    $('head').append(`<link href=\'
    }
  ],
  "index": 14
}
```

Copy the code below as a Control in your App Config. Make adjustments to the onSubDataRequest event for your specific usecase and render the co

Take a look at https://www.jstree.com/ for further configuration details and options.

Example JSON Structure from jsTree

```
// Alternative format of the node (id & parent are required)
{
  id          : "string" // required
  parent      : "string" // required
  text        : "string" // node text
  icon        : "string" // string for custom
  state       : {
    opened    : boolean  // is the node open
    disabled  : boolean  // is the node disabled
    selected  : boolean  // is the node selected
  },
  li_attr     : {}  // attributes for the generated LI node
  a_attr      : {}  // attributes for the generated A node
}
```

Slider

```
{
  "id": "mySliderControl",
  "controlType": "htmlTemplate",
  "widthClassMd": "col-md-4",
  "refersToConfigSource": "",
  "htmlTemplateContent": "<style>\r\n    .slidecontainer {\r\n\r\n        width: 100%;\r\n        /* Width of the outside contai
  "events": [
    {
      "name": "onSubDataRequest",
      "code": "$(app.getHtmlElement(\"mySliderControl\")).find(\".rangeSlider\")[0].oninput = function() {\r\n    app.setFieldVal
    }
  ],
  "index": 5
}
```

Create a brixxbox TextBox myOutput for the Value of the Slider

App Editor Properties

AddToAttachements

This setting controls whether the captured image should be saved directly as an attachment to the current record.

AppAltViewName

Alternative View Name

All Apps with an underlying master table will create a view bbv_{AppName} if you want an additional view with a special name, you can specify that here. The bbv_* view will be created in all cases.

AppAttachments

Attachments

When enabled, users can add files to the records of this app.

AppColor

App Color

You can choose a color theme for your app. This will be used in the single page tab headers for example.

AppConfigToStartName

AppConfigToStartName

Place the id of an app config here to embed

AppConfigToStartSourceField

AppConfigToStartSourceField

The field that should be submitted to the child app

AppConfigToStartTargetField

AppConfigToStartTargetField

The field in the child app that should get the Source Field Value

AppDiscussion

Discussion

When enabled, the toolbar button for record based discussions will be activated.

You will need to activate discussions for the master table in the table editor before using this.

AppHistory

History

This will activate the audit trail record history in the toolbar.

You will have to create the audit history for your mastertable in the table editor first.

AppIcon

App Icon

you can choose an app icon, that will be displayed in the single page tab headers, for example.

AppMailHistory

Email History

When enabled, this app gets a new toolbar button for the email history sidebar. This sidebar will show emails regarding the actual record.

AppMasterTable

Master Table

You can define the underlying table for your app, if it stores data. You can also create a new table.

AppParameters

App Parameters

App Parameters are Flags, that can be set by calls from the menu. You can react oto these flags in app events to change your app behavior.

```
if(app.parameters.includes("myFlag")){
    console.log("Parameter is set");
}
```

AppPrefix

App Prefix

Just like table prefixed, you could specify a prefix for all your controls in your app. Opposite to tables, this prefix is not mandatory

AppPreviewRecordId

App Preview Record Id

This is just a convenient way of loading a record at design time. It is specially useful, if you need a record to be loaded when designing your app, because some fields might be invisible when no record is loaded. This has absolutely no effect outside the app designer.

AppTitle

Title

This title will be translated and it can contain variables from four loaded record, that will be replaced in the title, once a record is loaded.

We used a @prefix in the past, that is deprecated.

```
    Order {id} {cordAddressId.adrName}
```

This will result in: Order 1 John Doe

if no '{' character is found in the title, brixxbox will add an {id} to your title to display the loaded record id.

If no app title is specified, the app id will be displayed for your app.

AttaDocType

Here you can select a document type for the attachment.

AttachementFilename

User defined file name for the attachment. This can be extended with values from fields of the current application. This is achieved with @controlId at any position of the text.

A user defined file name is always extended by the current timestamp.

AutoDisableOnEdit

AutoDisableOnEdit

This will disable a field, if a record id is set. In other words, the field is enable in create mode but disabled in edit mode

ButtonStyleClass

Icon Size

Icons inherit the font-size of their parent container which allow them to match any text you might use with them. With the settings, we can increase or decrease the size of icons relative to that inherited font-size.

BlockInputDuringEventExecution

BlockInputDuringEventExecution

if set, the brixxbox will block all user actions until the event has finished

CascadeCopy

CascadeCopy

If this property is set, a copy Operation in the app, that contains this Grid Control, will copy all Lines in the Grid for that Record. Say you want to copy an order record with all its orderlines.

CascadeDelete

Cascade Delete

If this property is set, a delete Operation in the app, that contains this Grid Control, will delete all Lines in the Grid for that Record. Say you want to delete an order record with all its orderlines.

ChartData

ChartData

The column from the sub data source for the Chart data

ChartData

ChartData

Thsi is a simple form to configure the chart. Just list the Columns of the sub datasource you want as lines (or bars, depending on your chart type). The lines will be rendered with default behavior and random colors.

ChartDataJson

ChartDataJson

This is a more complex method to configure the chart than chartData but offers a lot more possibilities.

See a List of properties, you can set per line, or bar:

```
[{
    "data": "OverallPrice", //the column to take for the line values from the sub dataset
    "color": "red", //line color
    "fill": "false" //do not fill the gap to the bottom line
},
{
    "data": "OwnPrice", //the column to take for the line values from the sub dataset
    "color": "blue",
    "backgroundColor": "yellow", //background color for the gap or the bar, in case of a bar chart
    "fill": "0" //means you fill the gab to the first (0) line.
}]
```

ChartLabel

ChartLabel

The column from the sub data source for the Chart label

ChartLabelAxesX

Chart Label for X Axes

This is a text, that will be used to label the chart x Axis

ChartLabelAxesY

Chart Label for Y Axes

This is a text, that will be used to label the chart y Axis

ChartType

ChartType

Choose from different chart types

ComboBoxEditButton

ComboBoxEditButton

This setting determines if a Combobox has a Edit Button next to it, to jump right to the selected record.

ComboBoxListButton

ComboBoxListButton

This setting determines if a Combobox has a List Button next to it, to select the value from a List instead of the Combobox itself. The List provides more features for filtering and searching than a Combobox . The selected Value from the list will be set in the Combobox on selection.

ComboboxKey

ComboboxKey

This is a fieldid from the sub data source that will represent the value of the combobox. In most cases this is "id"

ComboboxMinSearchLength

ComboboxMinSearchLength

Minumin Characters for a comboboxy to trigger the search. -1 for no search function at all

ComboboxMultiselect

Combobox Multiselect

Set if you want your users to select multiple values.

ComboboxValue

Combobox Value

This defines what will be displayed in the combobox. It can be a single column from the sub data source or a string with placeholders

```
adrName
```

```
{id} - {adrName}, {adrFirstName}
```

ConcurrencyControl

Concurrency Control

This is used to prevent users from overwriting records. If two users have loaded the same record, and both make changes, than the user that saves first, will commit his changes, but the second user will not be able to save after the first user saved. In other words, you will only be able to save a record if noone saved it between you loading it and your save attempt.

CreateMenu

Create Menu

A default menu Folder and Entry will be created. This is just a more convenient way than using the menu editor. You will probably use the menu editor at some point to move things around in the menu tree.

CssTemplateContent

CssTemplateContent

The css Content for the control

Data

Data specifies where this control gets its Data from. If no DataSource is set for the control, you get a List of table columns from the master table. If DataSource is set, you have to enter a Field from this DataSource (like a controlId if your DataSource is a Config)

DateTimeUtc

DateTimeBox UTC

If this Checkbox is activated. You will see the Date time in you local timezone but the value behind the control is a utc (GMT) time. setFieldValue will assume you set a utc time and will add the local timezone just for the display. getFieldValue will give you the utc time as well. So users in different timezones will see different values, but in the database there is just the utc time stored.

DecimalDigits

DecimalDigits

Set the number of decimal digits here

DefaultValue

This value is set when the app is opened or a new record is created. In the background, the brixxApi function setFieldValue is called with this value.

Example 1

```
1
```

Example 2

```
"Hello World"
```

Example 3

```
await app.sqlReadValue("readAddressName", { id: 1 } )
```

DevMode

Develop Mode

If the develop mode flag is set, this Version of the app will no be delivered to endusers. They will get the latest, non dev mode version.

DisableGridResponsiveness

DisableGridResponsiveness

This will disable the responsive layout of the grid and you will be able to scroll horizontally

EditAbsoloutGridHeight

Here you can set a fixed height for the grid. The specification is in percent of the complete screen height.

EditDefineUnit

You can specify the unit character here ($, € ans so on)

Enable

Enable

This will enable or disable the input.

GridAutoRefreshIntervals

GridAutoRefreshIntervals

You can provide a comma separated list of values for seconds. For each value the user will have the option to select it as an auto refresh value for reloading the grid content. Values above 59 seconds will be rounded and shown as minutes.

e.g. A list like "**30, 60, 900**" will generate a drop down menu with options like "**30 seconds, 1 minute, 15 minutes**".

GridColumnFilter

GridColumnFilter

You can create filter in all grid columns. You need to specify the column name and the type of filter you would like for this column.

The following types are available:

- **text**: Will show a text input field for the search criteria.
- **select**: A selection for all available options is shown.
- **multiSelect**: like **select**-option. It is possible to select multiple entries.

```
[
   {
       "column":"imImei",
       "filterType":"text"
   },
   {
        "column":"imItemId",
        "filterType":"select"
   },
   {
        "column":"imItemGroup",
        "filterType":"multiSelect"
   }
]
```

If the grid is configured for Server Side Paging additional options are available.

- **serverSideMinInputLength**: Allows to set a minimum character count for user input, before the search is triggered.
- **serverSideSQLStatement** : If the filter option is set to select/multislect a SQL statement can be used to show the available options. For config based grids this is optional (A distinct value for the column will be generated by default). For SQL based grids this setting must be provided. The setting needs to have the name of a valid SQL Statement defined in this app.

The SQL statement must return a list consisting of 2 columns: id and text. The id value will be used as the search criteria. The text value will be shown to the user as option. The parameter “columnSerachValue” will contain the current user input.

Example of a valid SQL statement:

```
DECLARE @searchLike nvarchar(100) = '%' + @columnSearchValue + '%';
SELECT * FROM (
        SELECT 'id1' AS id, 'text1' AS text UNION ALL
        SELECT 'id2' AS id, 'text2' AS text UNION ALL
        SELECT 'id3' AS id, 'text3' AS text UNION ALL
        SELECT 'id4' AS id, 'text4' AS text UNION ALL
        SELECT 'id5' AS id, 'text5' AS text
        ) AS ResultList
        WHERE text LIKE @searchLike
```

For columns with date controls (**DateBox** and **DateTimeBox**) a date range selection will be provided. The range selection enables the user to set a start and end date. Additionally the brixxApi call 'gridColumnConfig' enables setting of predefined selection ranges. Those ranges can be picked by the user without the need to use the calendar. This way commenly used selctions can be predefined for each individual grid column, if applicable.

Example setup for predefined selections (Note: The **The moment.js Library** is used to set start and end dates):

```
//Code should be placed as event "onAppStart"

//each entry within 'range' must have an array. First element is start date for

//selection and second the end date. The third array element is an optional string.
//If provided it will show up as tooltip for the selection.
const predefinedSelections = {'range':
    { 'Today': [moment(), moment()],
      'Yesterday': [moment().subtract(1, 'days'), moment().subtract(1, 'days')],
      'Last 10 days': [moment().subtract(9, 'days'), moment()],
      'This Month': [moment().startOf('month'), moment().endOf('month')],
      '2 years back': [moment().subtract(2, 'years'),
                      moment().subtract(2, 'years').add(1, 'months'),
                      '2 years back as start and 1 year and 11 month as end of selection']
     }
};

brixxApi.gridColumnConfig ('gridName', 'columnName', predefinedSelections);
```

GridColumnOrder

GridColumnOrder

A comma separated list of column, that should be displayed as the left most columns.

GridFilterVisibility

GridFilterVisibility

Set if the filter (search) field above a grid should be visible

GridFooter

You can create a footer for grid columns. The value for "column" specifies the target column.

The following parameter can be supplied:

- **“column”:** name for the target column
- **“select”:** will specify the Range for data selection. Possible values are **“all”** or **“visible”**. **“all”** is the default setting and will select all available data from this column. With **“visible”** only data matching a selection will be processed. If no selection is active, all data will be shown.
- **“action”:** will specify what action will be taken for the given data. Possible values are **“sum”**, **“avg”** (average), **“min”** (minimum), **“max”** (maximum). **“sum”** is the default.
- **“decimalDigits”:** If you are using a config based grid the decimal digit setting will be inherited from the column setting. Otherwise the default will be 0. If u want to overwrite this behavior this setting can be used.
- **“textLeading”:** Add a text in front of the value.
- **“textTrailing”:** Add a text behind the value.

```
//the following will give a sum for all data for the column "cordGrossValue"
[
   {
      "column":"cordGrossValue"
   }
]

//the following will give an average for the visible data within a selection. If no selection is activ all data will be processed
[
   {
      "column":"cordGrossValue",
      "select":"visible",
      "action":"avg"
   }
]


//the following will give one decimal digit for the value and add leading and trailing text. e.g. "Total: 123.4 $"
[
   {
      "column":"cordGrossValue",
      "decimalDigits": 1,
      "textLeading": "Total: ",
      "textTrailing": " $",
   }
]
```

GridInlineEdit

GridInlineEdit

Grid inline editing is not available at the moment

GridNoHyperLink

This allows you to set a list of combobox columns, that will not have a hyperlink, like the usual combobox column would have

GridPageSizeVisibility

GridPageSizeVisibility

Set if the page size should be choose able for users

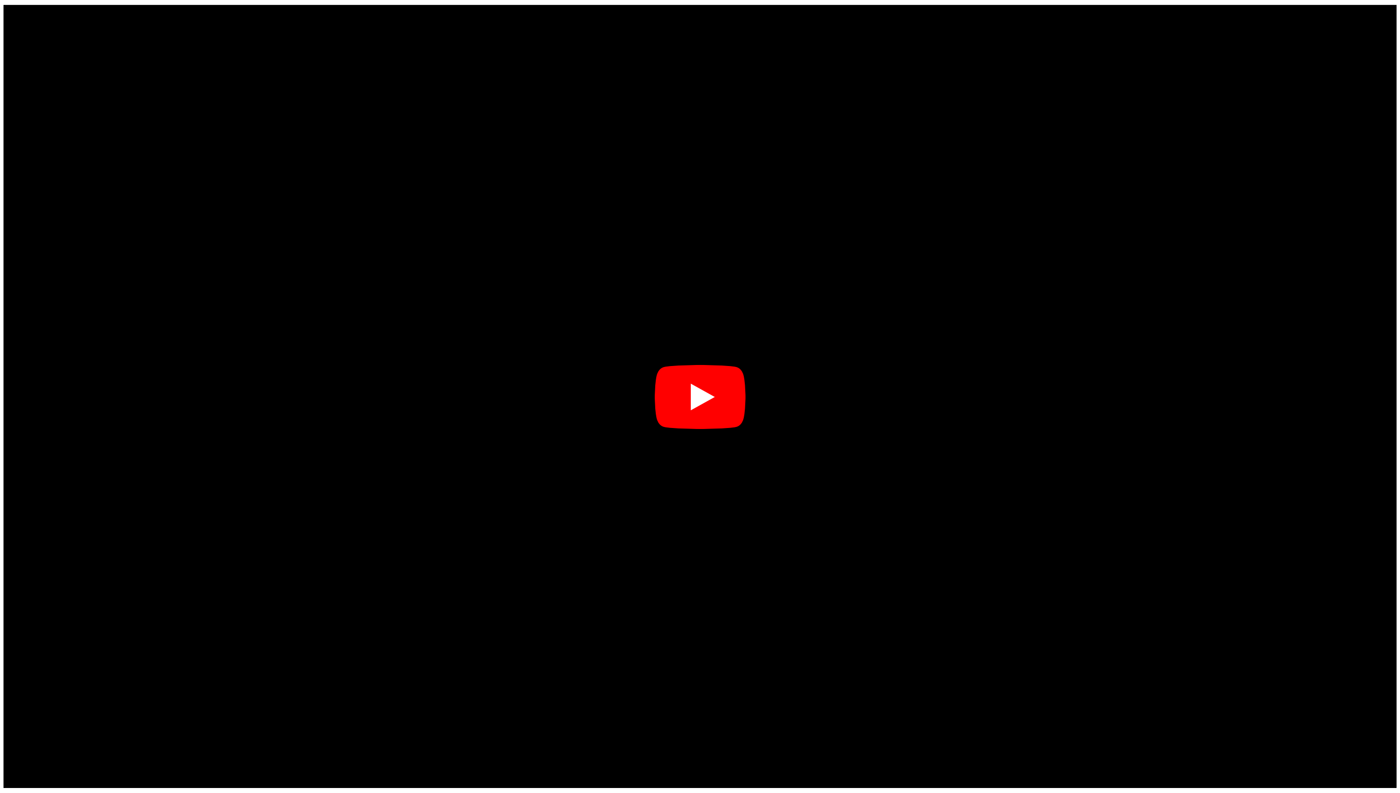
GridRowOrderDragDropParam

RowOrder Drag & Drop Parameter

If an app config is used as a sub data source, the values will be filled in automatically. If u want to set the parameter manually, the values must match the needed information for a SQL-Update statement. See the example below. In addition the relevant table/tables must be prepared to be capable to read and write the ordering data. See the list provided for Manual ordering of rows in a grid

```
{
   "targetTable": "exampleTableName",
   "targetKeyField":"exampleId",
   "targetRowOrderField":"exampleRowOrder"
}
```

```
-- e.g: Those parameters will lead to an update statement like this
UPDATE exampleTableName SET exampleRowOrder = @newRowOrderPosition WHERE exampleId = @id;
```



GridSelectMode

GridSelectMode

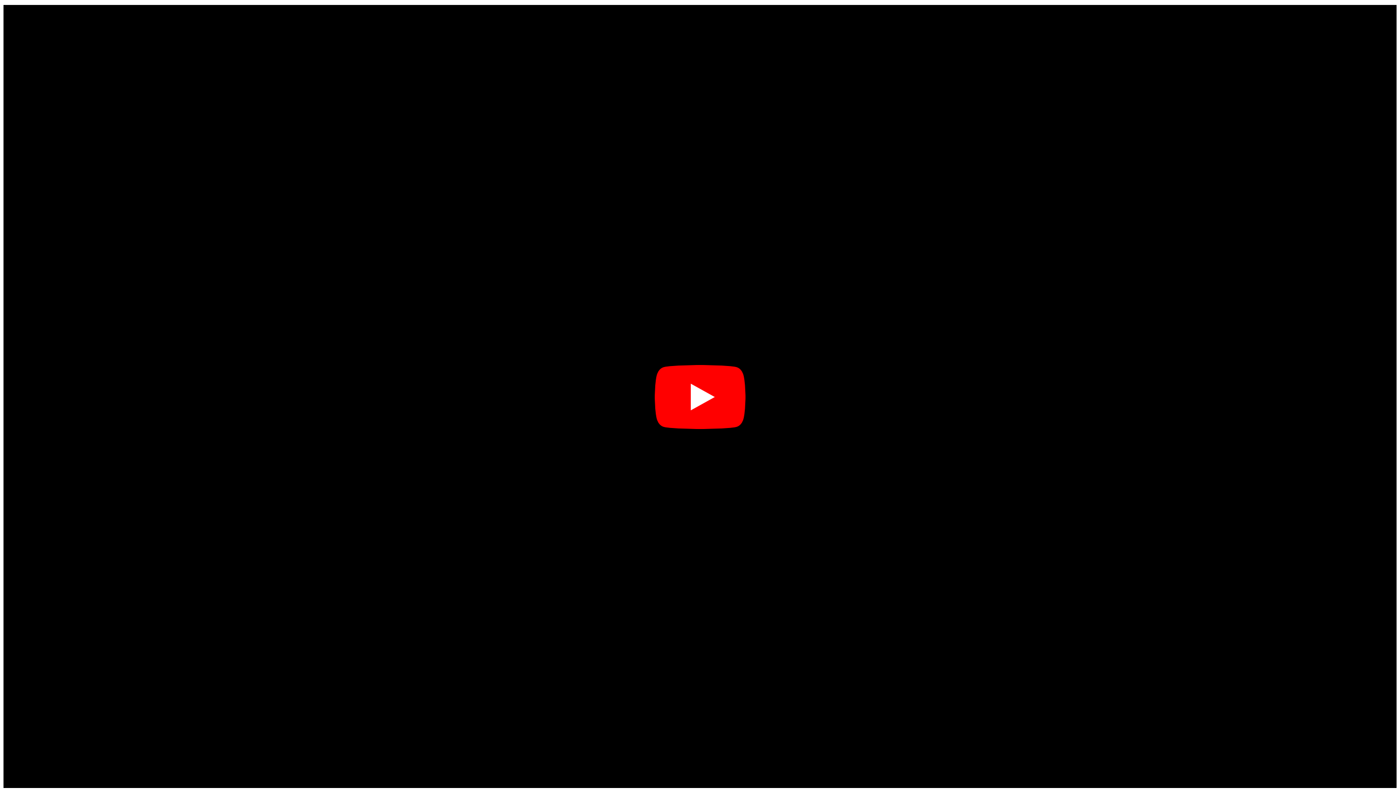
Choose the select mode

GridSortingMode

GridSortingMode

Set if sorting by the user should be allowed

If you want to allow the user to manually sort rows in the grid, options for "Allow Drag & Drop" are available. To use this extended option the underlying sub data source must be able to handle the manual sorting information. If an app config is used as a sub data source please check Manual ordering of rows in a grid before activating the setting. The app config only needs to be enabled for handling the necessary data. If you want to use a SQL-Statement as your sub data source the relevant table/tables must be prepared to be capable to read and write the ordering data.



GridToolbarVisibility

GridToolbarVisibility

Set the visibility for the toolbar right above the grid

GroupGridColumns

GroupGridColumns

A comma separated list of columns, that should be used for grouping. By default the grouping criteria will be the cell values for this column. The same cell values are put in one group. The column itself will be hidden from view because it will only show the group name. If a search is added to the column, the column stays visible.

For **DateTimeBox** the grouping value is set to the date part only and the column stays visible in order not to lose the time information.

For columns with date controls (**DateBox** and **DateTimeBox**) an individual grouping can be tailored to the specific data. This way date ranges can be shown in groups. e.g. "today", "this week", "last month". The brixxApi call 'gridColumnConfig' enables setting of this individual grouping ranges.

Example setup for date grouping (Note: **The moment.js Library** is used to set start and end dates):

```
//Code should be placed as event "onAppStart"

//each entry within 'grouping' must have an array. First element is start date for
//the group and second the end date. If groups overlab (e.g. "today", "this week", "this month")
//the first matching entry is picked.

columnAddOnSettings = {
    //add some grouping criteria for this date column
    'grouping':{
        'Future Dates': [moment().add(1, 'days'), moment('9999-12-31')],
        'Today': [moment(), moment()],
        'Last 7 days': [moment().subtract(6, 'days'), moment()],
        'This Month': [moment().startOf('month'), moment().endOf('month')],
        'This Year': [moment().startOf('year'), moment().endOf('year')],
        'Last Year and Older': [moment('1970-01-01'), moment().endOf('year').subtract(1, 'year')]
    }
};

brixxApi.gridColumnConfig ('testGrid1', 'cmpSomeDate', columnAddOnSettings);
```

HelpText

Help Text

You can set a help text, to display below a field.

HiddenGridColumns

Hidden Columns

This is a comma separated list of control, that you want to hide in this grid control.

```
    myControl1, myControl2
```

HideEditButtonColumn

hideEditButtonColumn

Set this if you want the edit button column to disappear.

HideSelectCheckboxColumn

hideSelectCheckboxColumn

Set this if you want the select checkbox column to disappear.

HtmlTableClasses

Html Table Classes

Possible classes

1.
    - table-bordered
    - table-striped
    - table-hoover
    - table-condensed
    - table-responsive

HtmlTableDisablePagination

Disable Pagination

If checked, there will be no pagination for the HTML table.

Be careful with the amount of data!

HtmlTemplateContent

HtmlTemplateContent

The html Content for the control

Id

The controlId is an essential part of a brixxbox Control. Every Control needs a unique (in its App) Id. All new controls get a generated Id (starting with an underscore). The generated Id is changeable but once you give your control a custom Id, this will be unchangeable afterwards. The generated Id should never be used to reference the control. If you need to do something with the control (like hide it, get or set a value), you need to give it a readable Id.

A Control is always addressed with its controlId.

Control Ids should start with a lowercase (often with a mandatory prefix, that has been set in the app) and can contain, Numbers, upper and lowercase letters and underscores.

IncludeInGlobalSearch

If this is checked, this column will be included in the global search scope

InputMaskRegEx

1. Input Mask Regex

You can mask the uinput of a textbox by using a regular expression.

You can find further information for for the Input Mask itself [here](https://github.com/RobinHerbots/Inputmask)

If you want to lear more about regex, take a look [here](https://regex101.com/)

Examples:

This allows only non-capital and capital letters (azA-Z) numbers (0-9) and the underscore (_). But no special characters (like ö, &, %, @,...).

```
^[_0-9a-zA-Z]*$
```

Example 2. This forces the input to begin with abc, followed by any a-Z0-9

```
^abc[_0-9a-zA-Z]*$
```

InputTag

InputTag

if set, the user is allowed to add new tags

Label

This is the Label, that is displayed for this control. The Label is translated automatically but you can set your manual translation in the translation Tab.

All Labels must be in the Workspace Default Language (english if not set differently in the workspace settings)

LabelWidthClass

The Label With is only relevant if the control is in a From Group, and so the Label is left or right of the control instead of above it.

It uses the same rules as the Width property

Please have a look at Bootstrap Grid System

LabelWrap

If this checkbox is checked. the label text will be displayed in full length and wrapped at the end of the available width. If not checked the label will be cut and a "..." will be appended. You can hover with the mouse over a label to get the full text.

LoadRecordOnLeave

If this is set. Brixxbox tries to load a record by using the value of this control as a key.

ManualRefreshOnly

No automatic refresh

This control is excluded from app.refresh() calls if the control name is not explicit mentioned.

app.refresh("myGrid") will refresh it but app.refresh() will not.

Use this setting if the grid data is expensive to get and you want to avoid unnecessary calls for data

NoSpinButton

Check this option, if you do not want up/down arrows next to the number input.

RecordObservation

Record Observation

When 2 ore more users open the same record, only the first will be able to edit it. Other users will have to option to request the edit mode.

RefersToConfig

If this is a field, that other fields refer to, to get data from another Config. You have to specify which config that is.

ReportCulture

Report Culture

The report culture determines e.g. how date values are displayed.

ReportId

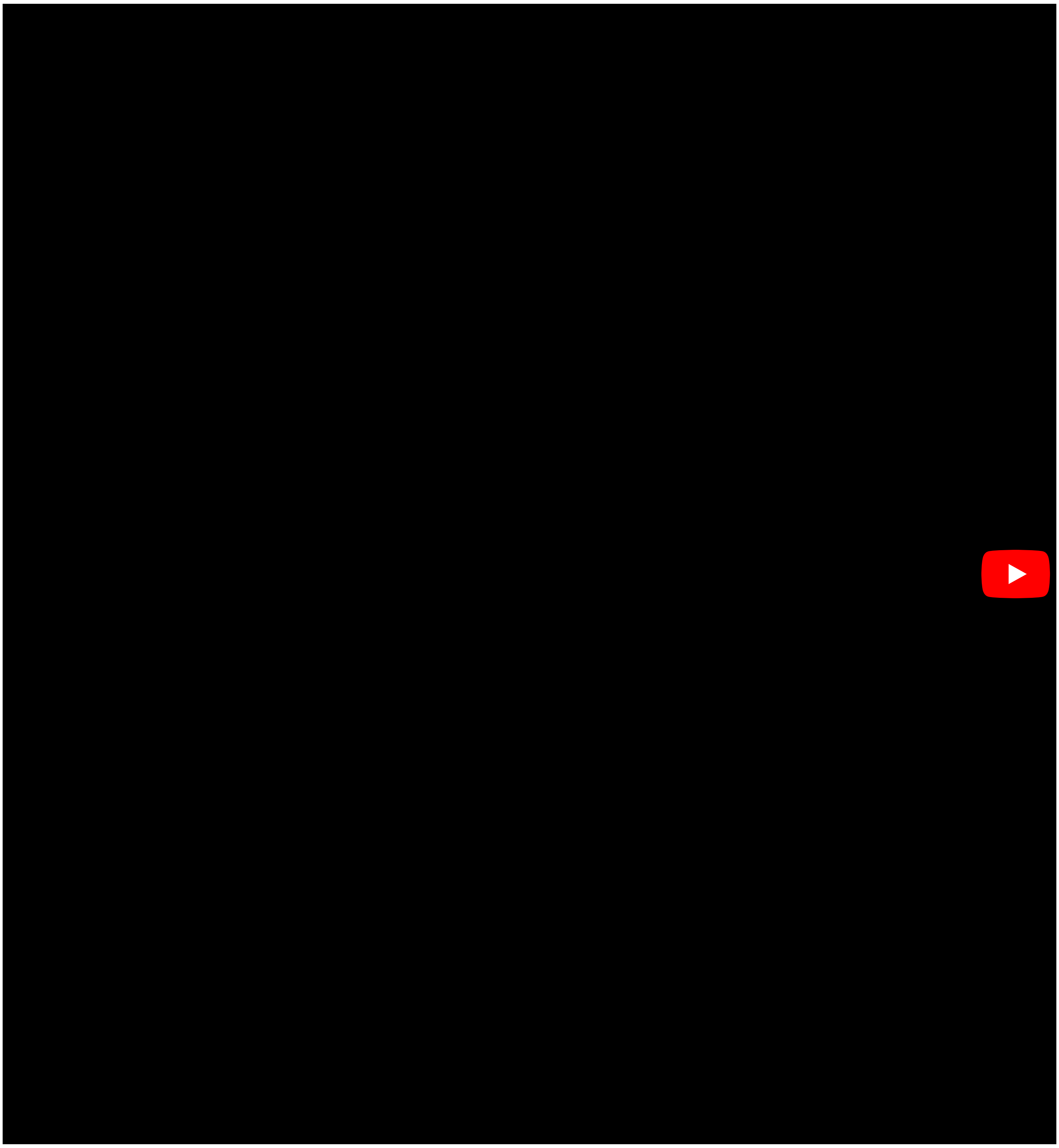
Report Id

Choose a report to use in this Report Control. Reports can be managed in the Report Config App

RowReorder

Manual ordering of rows in a grid

This option will prepare the app for manual ordering of rows in a grid. Only set this option, if you want users to be able to order grid rows by drag and



Setting this option will make the following changes to your app:

- **Create a new field** for the underlying datatable. The default name will be **brixx_RowOrder**. Type will be set to **HierarchyId**.
- The **View** for this app will get a new column to show the RowOrder with hex value.

```
CONVERT([varchar](512), CAST([ExampleTableName].[brixx_RowOrder] as varbinary), 2) as brixx_RowOrder]
```

- A **InsertTrigger** will be created for the new field. Every time a new row is inserted without a value for brixx_RowOrder an order value will be gene

```
CREATE  Trigger [dbo].[brixx_ExampleTableNameInsert_RowOrder]
ON  [dbo].[ExampleTableName]
AFTER INSERT

AS

SET NOCOUNT ON;

/* IF all brixx_RowOrder fields are valid, no need for any processing */
IF (SELECT COUNT(*) FROM inserted WHERE brixx_RowOrder IS NULL) = 0
BEGIN
        --PRINT ('No Action needed for Trigger on ExampleTableName');
        RETURN;
END

DECLARE @RowId BIGINT;
DECLARE @InsertAfterHierarchyId as HierarchyId;
DECLARE @ParentHierarchyId as HierarchyId;

DROP TABLE IF EXISTS #tmpTable;
SELECT Id INTO #tmpTable FROM inserted WHERE brixx_RowOrder IS NULL;

WHILE EXISTS (SELECT * FROM #tmpTable)
BEGIN
        SET @RowId = (SELECT TOP(1) ID FROM #tmpTable ORDER BY Id);

        /* If the INSERT does not provide a HierarchyId, generate a value at the end of list */
        SET @InsertAfterHierarchyId = (SELECT TOP 1 brixx_RowOrder FROM [dbo].[ExampleTableName] WHERE brixx_RowOrder IS NOT NULL
        IF (@InsertAfterHierarchyId IS NULL)
        BEGIN
            -- Default Parent ID
            SET @ParentHierarchyId = '/1/';
        END
        ELSE
        BEGIN
            -- Take Parent ID from Child
            SET @ParentHierarchyId = @InsertAfterHierarchyId.GetAncestor(1);
        END

        UPDATE [dbo].[ExampleTableName] SET brixx_RowOrder = @ParentHierarchyId.GetDescendant(@InsertAfterHierarchyId, NULL) WHER

        --PRINT ('INSERT Trigger on ExampleTableName. Parent: ' + @ParentHierarchyId.ToString() + ' Insert After: ' + @InsertAfte

        DELETE #tmpTable WHERE Id = @RowId;
END

DROP TABLE #tmpTable;
```

- If no **ORDER BY** is provided for the SELECT Statement, ORDER BY will use the brixx_RowOrder Field
- **All Existing data will be updated!** If rows exist when saving your app brixx_RowOrder will get a default value for ordering.

```
UPDATE ExampleTableName SET brix_RowOrder =  '/1/' + CONVERT ( nvarchar(30) , Id) + '/' FROM ExampleTableName WHERE brix_RowOrder
```

SearchPrefixes

Search Prefixes

If a field is included in the global search, the prefix can be used to search for specific records only. You can specify a list of prefixes for every field that is included in the search. Lets say you have an address with a company name, a first name and a last name, you could give the following prefixes:

- Company: address, company
- FirstName: address, name, firstname
- LastName: address, name

You could search for "address" and search all the fields, or you could search for "name" and look only in first and last name

SelectAllOnFocus

Selects all text when the focus is set to this control.

ServerSidePaging

Server Side Paging

Reads only the actual page from the server instead of all table data at once. Use this for large tables to improve performance.

The subdata requests must NOT have ORDER BY

ServerSidePagingIndexColumns

Server Side Paging Indexed Columns

When using serverside paging, most of the time it is not usefull to search in all columns as that means the grid is as slow as if it was not paged on server at all. We need to configure the columns that are indexed and can be used for ordering and searching. The other columns will not get a order option and will be ignored for searching. In case of a combobox (cordAddressId) the _value column will be searched as well. no need to configure that.

```
adrName, adrFirstName, adrCompanyId
```

SmoothScroll

Smooth Scroll

Lets the user scroll through the table instead of using the paging buttons

TagValue

TagValue

The value that should be displayed

TextColor

Text Color

Choose a text color

ToolTip

Tool Tip Text

Enter a text for a tool tip. This will be translated.

ValidateInputBeforeExecution

ValidateInputBeforeExecution

If set, the brixxbox will check the validation before executing the event sourcecode

ValidatorMessage

ValidatorMessage

You can enter a message here. In most cases that is not necessary, as there are default texts for all validators

Visibility

Visibility

You can specify the default visibility of a field

WidthClass

You can specify your display layout for 4 different categories of devices.

- **extra small** - screens less than 768 pixel wide like **phones**.
- **small** - screens less than 992 pixel wide like **tablets**.
- **medium** - screens less than 1200 pixel wide like **small laptops**.
- **large** - screens with more than 1200 pixel wide like **laptops and desktops**.

You can make a setting for each of those categories. Or u can just skip the layout for a setting and the smaller setting will come into place. The screen width is always split into 12 logical columns. The smallest line width to choose is 1/12. brixxbox will set the extra small device to 12/12 as default. An input control will go across the full screen width. Medium devices will be set to 4/12 per control. That will lead to 3 controls per screen line for medium and large layouts.

In addition you can decide if u want to have the control label on a separate line above the control or in front of the control. Those settings will be used for all devices.

After saving the control settings in app designer the preview will reflect the changes.

For further information please have a look at Bootstrap Grid System

WysiwygHtmlCleanup

When copy pasting HTML code (from word or other pages) to the WYSIWYG editor, the HTML can be pretty blown up and this can lead to issues printing the to complex html text in telerik reporting.

This setting allows to extract the plain text, when pasting into the editor. All the unsupported html tags are removed but you have to do the formatting again in the editor.

WysiwygHtmlFragment

When checked. brixxbox will save the html content as a fragment, exactly as you see it in the editors code view. If unchecked, brixxbox will wrap the content in a valid html document with header tag and <body></body>.

If you would like to print the text in telerik reporting, you should check this, to keep the whitespaces.

Controls

grid

A grid displays a list of records, specified by the sub data source of that control.

Documentation

In Brixxbox, app configuration allows you to add different controls in your app. It also provides four types of properties which manages how control should behave or look in app. Those four property types are:

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are the main properties of any control. These properties define how control behaves in Brixxbox App.

- Control Type

- For using grid control, click on "Add Control" and select "Grid" control type from drop down.

- Control Id

- For each Brixxbox app, this id should be unique. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value but it should be meaningful. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters.

- Label

- It is the display value for control.

- Refers to Config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Hide Select Checkbox Column

- By default, grid also display's a checkbox with each row. This checkbox is used to select a specific row. If it is enabled then user can select multiple rows at a time and perform different operations on them like deletion. If user does not want to display this checkbox then this property should be checked.

- Hide Edit Button Column

- This property is by default unchecked. It allows grid to display an edit button with each displayed row. If user does not want to display Edit button then this property should be checked.

- Cascade Delete

- This is a checkbox. If it is set and a delete operation is performed on a record, in an app containing this Grid control then all the lines of a record will get deleted. For example: User wants to delete an order with all of its item's lines.

- Cascade Copy

- If it is set and a copy operation is performed on a record, in an app containing this Grid control then all the lines of a record will get copied. For example: User wants to copy an order with all of its item's lines.

- Server Side Paging

- Server Side Paging

- Use this property for big data tables. If checked, grid will only be able to data from server side only instead of getting all the data from data source.

- Smooth Scroll

- If checked, this property allows the users to scroll through list of items of grid instead of scrolling up and down via paging buttons.

- No automatic refresh

- Use this property when fetching expensive Grid data. If set, it will block unnecessary calls for data. Brixxbox allows an app to refresh data of all of its controls using "app.refresh()" function. Settings this property will exclude this grid from data refresh calls. However, you can refresh grid data by specifying the grids name in refresh function. For Example: app.refresh("myGrid").

Tutorial

Gird control can be used in two possible scenarios: displaying all data of sub data source. For example: listing all customer orders, and displaying some values. For example: listing order lines of a specific customer order only.

This tutorial assumes that a user has an existing customer order app which records customer's order data. They want to list all customer orders in a new app and update grid to display all order lines of a specific customer order in customer order app.

For first scenario, create a customer order list app, select "Grid" from list of controls. After that you need to give an id to your grid. As grid is usually used to display data from a sub data source, so there is no need to create column in database for Grid control.

Let start with the first case. Select a new app, name it "customerOrderList". Select a "Grid" control type from the list, assign a unique id and meaningful label:

As you can see from the above picture, we have added a grid control but nothing appears in app editor on right side. This is not an error but expected behavior because grid control is used to display a list of records but first we need to specify "sub data source" in grid properties. For this click on "edit sub data source" and select "config" as a data source type. Now select "CustomerOrder" app to display all customer orders and save app settings.

Now you can see that our grid list all customer orders. In its menu, grid allows multiple options like adding new data row, deleting data row, refresh grid, copy data, search filter and drop down list for specifying number of entries for display. Grid provides each data row with plus, select, and edit button. Depending on screen size grid displays only few data columns and plus button is used to see remaining columns as shown below:

Grid allows us to enable or disable select and edit buttons(see General Properties). We have successfully listed all customer orders, now we move towards second scenario. We have two relations between customer order and order lines which are: customer order can have multiple order lines, and one order line can only belong to one customer order. In this part, we want to cover second relation. Now we want to display order lines which belong to only one customer order in customer order app grid. Our customer order app looks like this:

There are a lot of details in this app which are not important here. Our concern is order lines gird which will display only those order lines which belong to one specific order. To allow this behavior, we need to edit "sub data source" settings of our order lines grid and set "target field binding" with the customer order line id.

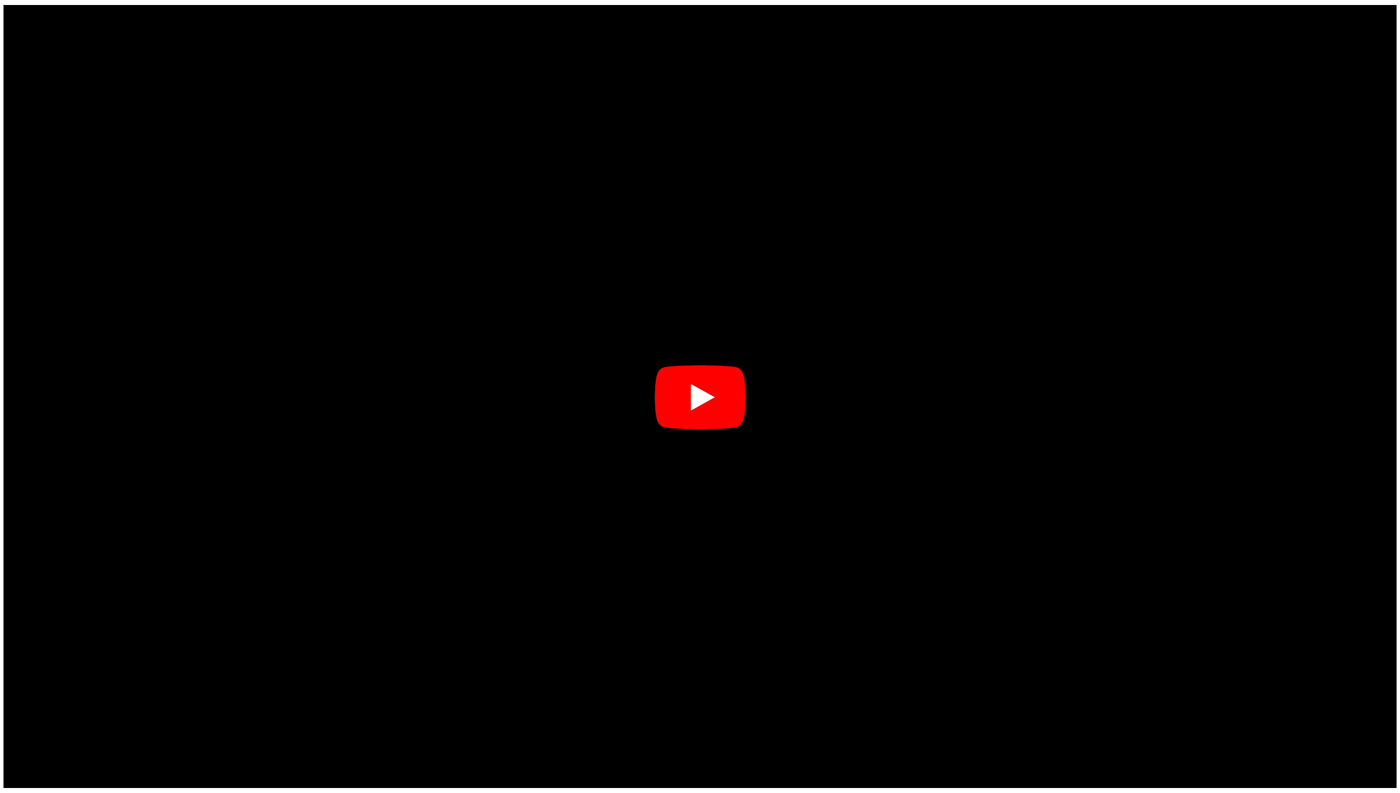
Now we will see only those order lines in our grid which belong to same customer order. Lets add a new order for our customer John and add a new order line for this order.

Now order lines which belong to John's order are being displayed in our order lines grid. In our case we only added one order line.

combobox

Comboboxes are often used to refer to header records (an order line record will most likely have a combobox with its order record).

Demo



Documentation

In Brixxbox app configurations you can add different controls. One of them is ComboBox. Each control has four types of settings

1. General Properties
2. Size
3. Style
4. Translation

General Properties

These are main behavioral properties of any control. These are responsible for control's functionality.

- Control Type

- For using combobox control, combobox control type should be selected from drop down list.

- Control Id

- For each Brixxbox app, this id should be unique. By default Brixxbox assigns a random id which starts with underscore and followed by a string. For Example: "_abcdef". Brixxbox allows you to change its value only once. You can change this id to any value. Recommended way is to start with mandatory prefix(set in app). It can contain numbers, underscores, uppercase and lowercase letters. Each control is always accessed with its Control Id. For Example, you can get and set control value by using its ControlId. If you want to store value of this control in database, use the database button to create a column. You will be able to select column datatype, max length, can be null, options.

- Label

- It is the display value for control.

- Data

- This property specifies from where this control gets data. If no option is specified then by default "Not in Database" option is chosen. If you want to set it then you have to specify the controlId of data source.

- Refers to config

- You need to set this option if this is a field, that other fields refer to, to get data from another config.

- Combobox Key Field

- This is the field value from the sub data source. It also becomes the value of our combo box. When we have to get the value of our combo box using "getfieldvalue" function. It will return the corresponding id of selected combobox entry.

- Combobox Value Field

- This option defines how each entry of your combobox will be displayed. It can be a single or multiple columns from sub data source. For single column just name the column. For Example, "adrName". If you want to choose multiple columns, then use curly brackets. You can choose any place holders between them. For example, {id} - {adrName}. "-" is a place holder while id, adrName are column names.

- Min Search Length

- This option specifies the minimum search length of string to trigger search function. If you do not want any search function then specify "-1". In case you want to specify after how many characters Brixxbox should trigger the search and get actual records from the sub data source, specify it here as positive integer. For Example, if value 3 is specified then Brixxbox will trigger actual search only after 3 characters are entered in search field.

- Multiselect

- When this option is specified, one can select multiple values as combobox entries.

- Select List Button

- By default this is hidden. It opens a list of all items of sub data source. It can be used to select an entry from combobox.

- Select Edit Button

- By default this is enabled. If a value is selected in combobox then by clicking edit button will open the complete record in corresponding application. If it is clicked without any value it will open the corresponding application with no record. It can be used to add a new record.

- Default Value

- This value is set when an application is opened or a new record is created.

Tutorial

This tutorial will follow the similar example used in the demo for combobox. You can find this demo at the top of this page.

This tutorial assumes that a user has an existing app and they want to add a new control of type "Combo Box". First of all, select "combo box" from list of controls.

After that you need to give a id to your combobox and create column in database if necessary. Here we named it "myCombo". Also, add a public label for combobox. We labelled it "Address".

Now save the control. It is time to assign a sub data source to our control. Click a sub data source button. Here you can assign different types of data sources. For this example, we will use config data source type. After that select the corresponding app config. We are selecting "address" config because we want to display all the addresses.

Up till now we have defined a combobox, assigned a unique control id and also added a sub data source to it. Now we want to assign two main properties to combobox: combobox key field, combobox value field. "Combobox key" property is a column from result set that will also be the value of our combobox when the entry of combobox is set. Mostly it is the id of result set. Here we also choose id. We can use this value for comparisons. For combobox value field, we want to display values of two columns: adrNumber, adrName from result set. We have to use curly brackets when using multiple columns.

Each entry of combobox will now represent the number of the address and the person's name. For example "10001 John". In this way we can define and use combobox but can we use the value of combobox to display or change some other data? the answer is yes!. For demonstration, we will add grid control (see grid documentation) "myGrid". We also assign address config as a sub data source for our grid control. Now "myGrid" control will display all the addresses present in the result set.

As you can see from above picture that nothing is selected in our combobox "Address". Now we want that whenever we select a record in combobox, the grid displays the same selected record only. For this purpose, we need to modify sub data source of mygrid and add a where clause. An important thing to note here is that all controls of an app are available as sql parameters. In where clause, we need to specify a condition which displays only a record which matches combobox value. We can specify this as "id=@myCombo". Remember that we selected "combobox key field" as id of sub data source "address". In order to pass this change to grid, we have to add a control event "onChange" to our combobox control. Whenever our combobox value change, we want "myGrid" to refresh.

Now save and set a combobox value and we will be able to see only selected record in "myGrid". As we can see from figure below.

Combobox also allows us to select multiple records at a same time. To achieve this functionality, select "multiselect" checkbox from combobox properties. You can see from figure below that multiple values are selected in combobox but there is no record shown in grid.

This is because we were using where clause for comparing one value of combobox but now our combobox has multiple values. Now if we call "getfieldvalue" function on our combobox. It will return an array with selected combobox key value's. Now we have to update our where clause logic in grid to display all selected records. We add this statement to where clause **"id in (select value from OpenJson(@myCombo))"**. This statement says that if a value is present in result set then display it in grid.

Template Gallery

Extended CRM-System

Contents

What is CRM

Customer Relationship Management (CRM) is a strategy for companies and their employees to systematically create relationships and interactions with contact persons of existing and potential customers. By generating sales, these customers play a decisive role in the long-term success of the company. If customers adjust their buying behavior and switch to products from other companies, this can have serious consequences for their own company. That's why companies invest a lot of time and effort in CRM activities to analyze customer preferences. Target groups, markets and the respective needs are analyzed within the CRM and coordinated with the own portfolio. If deviations are found, the employees of the sales and marketing department, in coordination with the management, have the task of optimizing the customer relationship within the framework of CRM or adjusting the concrete CRM strategies in order to better reach the customers again. Other departments also contribute to customer satisfaction and provide valuable information for CRM. This includes, for example, the entire service department. Within the scope of service, customer inquiries and discrepancies are recorded and processed. This is also part of CRM.

CRM generates a wide range of detailed information that must be bundled, categorized and evaluated in order to be able to derive statements. Without a corresponding technology, it does not take long with increasing numbers of customers and prospects and the respective contacts, until CRM can no longer be managed with manual work. Targeted contact with the customer is made more difficult. Without a suitable technology, employees lose too much time for CRM if the CRM strategy is to be pursued with the same quality and intensity even with increasing numbers of customers and prospects.

What is a CRM-System

A CRM system is a software with functions for the management and automation of all information that is generated month by month when addressing customers or prospects and binding them. In the CRM software, all data of customers and prospects, including all transactions and related communication, are collected and stored in databases. The following information can be taken as an example:

- Documentation of a prospective customer telephone call for later evaluation
- Storage of a customer offer for later allocation
- Appointment reminder for an offer meeting with an interested party on site
- Birthday management of A and B customers to increase attention

CRM systems are mainly based on standard software products. Depending on the size of the company and the required level of information, different solutions with different functionalities are possible. If a company grows over time, the requirements and thus the necessary functions of the CRM system also grow. It can therefore happen that the previous functions of the CRM software are no longer sufficient over time and a change must be considered in order to take the increasing degree of information for the company into account.

Advantages and benefits of a CRM-Systems

The advantages and benefits of CRM software for companies are obvious. In practice, a large number of contacts and their relationship to the own company must be managed in one system. Well-designed CRM systems create more transparency. They can be used wherever direct or indirect customer or prospect contact is required. With the right CRM software, the entire sales force can also be supported. This applies both to:

- the sales force with the objective of "increasing sales",
- as well as for the service-oriented field service with the objective "service and maintenance".

All activities on site can be recorded directly in the CRM software. In addition, a good CRM system can also be an aid for certain activities, in which information is available at the right place in the right form. This not only saves time, but also regularly delivers better results in all areas. In summary, a CRM system helps to support sales, marketing, service and last but not least, management. With the help of a CRM system, activities can be optimized and automated (e.g. monthly reports). In addition, a good CRM provides a variety of decision bases.

Advantages and benefits of an integrated project management system

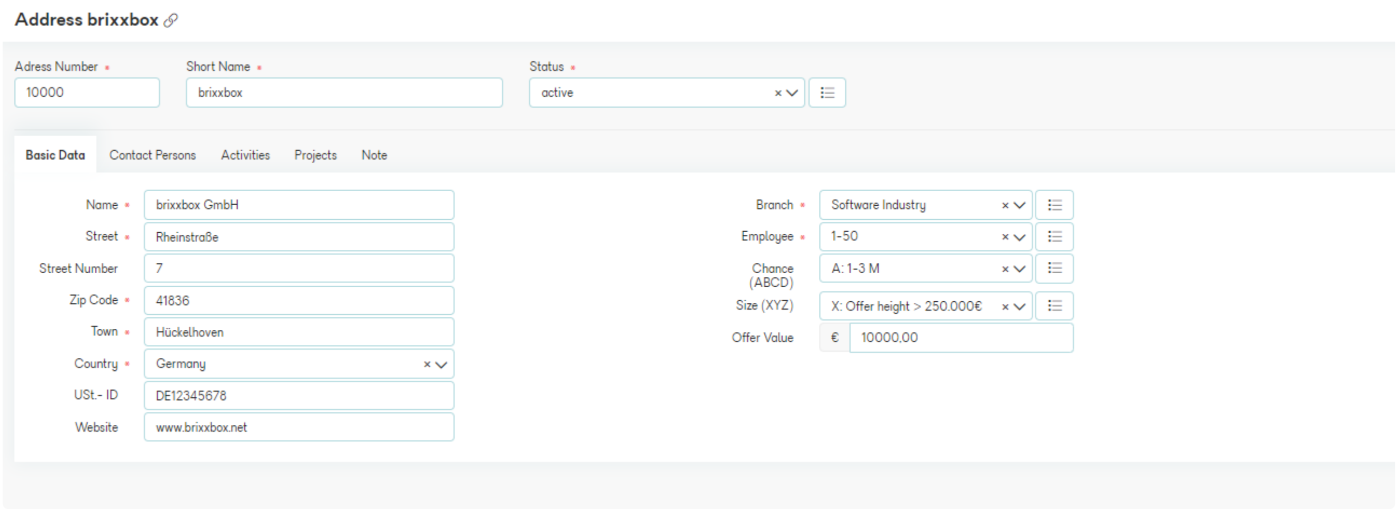
Projects are regularly associated with investments and other expenses and are directed towards a specific goal (project objective). In practice, a project is often divided into sub-projects, each of which is aimed at a specific (partial) goal fulfillment and is related to the main goal. The project management is responsible for planning and controlling the (sub-) projects, as well as monitoring them with regard to time- and resource-oriented parameters. As with CRM, project management involves strategies that must be supported by suitable technologies and systems at a certain level of complexity at the latest in order to process the wealth of data in a meaningful way. A CRM system with integrated project management saves time and effort and increases data quality.

The sales department acquires a new customer project as an example. Up to now, all customer- or prospect related activities were recorded in the CRM system. The following project is now regularly managed by other employees and departments. Thanks to the integration, project-specific information can now be found in the shortest possible time. The management of the information is clearly optimized by the symbiosis of both systems.

brixxbox Low-Code-Plattform: Extended CRM-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.


Address brixxbox

Adress Number *	Short Name *	Status *
10000	brixxbox	active

Name *	brixxbox GmbH	Branch *	Software Industry
Street *	Rheinstraße	Employee *	1-50
Street Number	7	Chance (ABCD)	A: 1-3 M
Zip Code *	41836	Size (XYZ)	X: Offer height > 250.000€
Town *	Hückelhoven	Offer Value	€ 10000,00
Country *	Germany		
USt.- ID	DE12345678		
Website	www.brixxbox.net		

In practice, this CRM application could be used for example by:

- Consulting company
- Freelancers
- and small enterprises

to create customers and prospects and manage their contacts and CRM-relevant data. In addition to master data management, advanced functions are also included to visualize the possible use and linking of master data and to derive a realistic practical reference to support management, sales and marketing.

Extract of the functions

- Company data: Name and address of the companies
- contact details: Contact data and contact persons of the companies and employees
- Phone book: based on contact data

- Archiving: storage and digitalization of documents and files from customers and prospects
- Special features: Documentation of specific customer requirements and special features (e.g. billing only per month)
- CRM data: Scheduling (e.g. per month, week), deadline monitoring and appointment documentation as well as assignment of relevant contacts
- CRM Workflows: Creation of reminders and follow-ups (e.g. in two months) for certain contacts
- CRM report: e.g. ABC/XYZ - evaluation or categorization of customers and prospective customers (periods/months)
- Templates: text templates can be created
- Project integration: project planning, project monitoring, resource management and responsibilities
- Management: Status monitoring of all activities and projects
- Mobile: Access to all data anywhere and with any device (cloud)

Time Recording

Contents

Why do I need a time recording:

In many companies, the principle of confidential working hours was practiced for a long time for some employees. This is an organizational model in which the completion of tasks is in the foreground. Not the working time, but the work result is decisive. This model was mainly practiced by creative working employees, since a rigid working time schedule, also from a scientific point of view, hinders creativity and thus above all possible innovations.

However, the European Court of Justice decided in 2019 that in future, the recording of working hours in all companies must be carried out by all employees. The ruling will oblige employers to offer a time recording concept that allows the entire working time as well as breaks and days off due to vacation or sick leave of the employees to be documented. This is to ensure that the laws regarding working hours, breaks and days off as well as the legally required vacations are adhered to for the protection of the employee.

When recording and documenting the working hours of all employees, a lot of information is quickly generated which must be bundled, categorized and evaluated to ensure compliance with the prescribed laws. Without supporting technology, it does not take long before time recording can no longer be handled by hand.

What is a time recording system:

A time recording system is a software with functions for the documentation and evaluation of all time-oriented parameters of the respective employees as required by law. In the time recording software, all relevant employee data is stored in databases. This includes the following areas in particular:

- Working hours
- Break times
- Vacation days
- days off due to illness

Time recording systems are mainly based on standard software products. Depending on the size of the company and the different working time models, such as shift work or piecework, different solutions with different functionalities come into question. If a company grows with the time, the requirements and thus the necessary functions of the time recording system also grow. It can therefore happen that the previous functions of the time recording software are no longer sufficient over time and a change must be considered in order to take into account the increasing level of information for the company.

Advantages and benefits of a time recording system

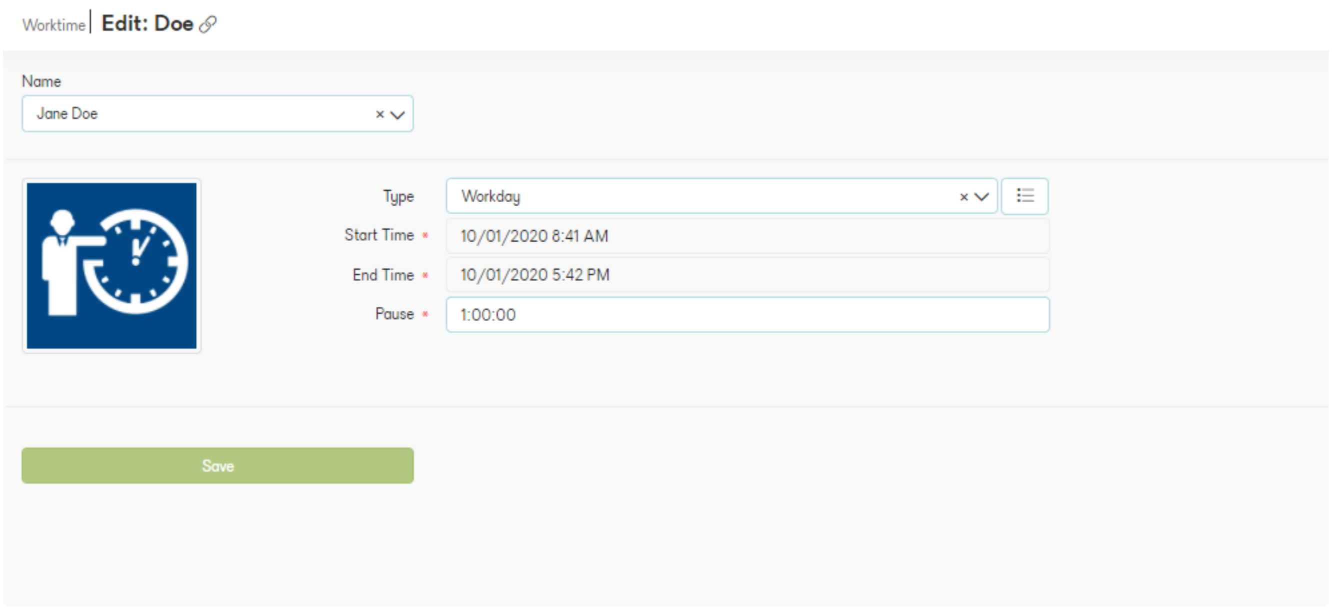
Good time recording systems offer enormous flexibility in terms of documenting the working hours of the workforce. There is a wide range of recording options, depending on the software used. For example, there are systems that allow input via cell phone. This is especially interesting for field service. In this way, data can be entered into the system quickly and without much effort. Of course there are also other end devices that are compatible depending on the system. These include, for example, terminals with chip cards or comparable systems that enable the employer to book the respective time types by simply hanging up the phone.

Furthermore, a good time recording system can also be an aid for certain decisions of the management or the personnel department and last but not least, a time recording system regularly provides the basis for payroll accounting.

brixxbox Low-Code-Plattform: Time Recording-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.


Worktime | Edit: Doe

Name: Jane Doe

Type: Workday

Start Time *: 10/01/2020 8:41 AM

End Time *: 10/01/2020 5:42 PM

Pause *: 1:00:00

In practice this time recording application could be used for example by:

- freelancers
- and small enterprises

to document their own working hours and plan your annual vacation.

Extract of the functions

- Employees: Employee administration with the corresponding contact data
- Recording: recording of the daily working hours and breaks of the respective employees
- Planning: documentation of vacation requests and status management of the respective employees
- Status: documentation of approvals

You can find further information at https://brixxbox.net

Project Management

Contents

What is project management:

The environment of a company is subject to permanent change. In certain phases, legal framework parameters change or new technologies lead to changed customer needs. Even if these are only excerpts from a complex overall structure, they still regularly have an impact on the success of a company. New tasks arise. Original goals have to be questioned and adapted from case to case in the context of projects and derived tasks. By definition, a project is a one-time (business) process, which is divided into different phases and tasks and is directed towards a specific goal achievement and has a fixed start and end date.

In this context, goals have to be defined, ways to reach them have to be found and managed, and project participants have to be monitored as a team and resources. The bundling of these activities is generally understood as project management. In practice, a project is often divided into sub-projects, each of which is aimed at a specific (partial) goal fulfillment and is related to the main goal. The project management is responsible for planning and controlling the (sub-) projects, as well as for monitoring them with regard to time- and resource-oriented parameters (e.g. the members of a team).

Project management generates a wide range of detailed information, which must be bundled, categorized and presented to the management for evaluation using appropriate methods in order to derive statements and make decisions with regard to the achievement of objectives. Without a corresponding technology, it does not take long from a certain level of complexity until project management can no longer be managed by hand. Without a suitable technology, the members of a team lose too much time for project management and the achievement of objectives is endangered.

What is a project management system:

A project management system is a application that provides the project members of a team with computer-aided assistance in their project activities. In the project management system all data and information of the project including the existing sub-projects as well as all transactions and the associated communication are recorded and stored in databases. The following information can be taken as an example:

- Documentation of a meeting for the target definition of individual sub-projects
- Budget management per (sub)project
- Scheduling of individual project steps

Project management systems are mainly based on standard software products. Depending on the size of the company, the respective team and the required level of information, different solutions with different functionalities are possible. If a company grows over time, the requirements and thus the necessary functions of the project management system also grow. It can therefore happen that the previous functions of the project management software are no longer sufficient over time and a change must be considered in order to take the increasing degree of information for the company into account.

Advantages and benefits of a project management system

The self-organization of a team or even an individual is an enormous challenge as the level of information increases. Handwritten lists, here and there a piece of paper or a post-IT are of only limited help and are certainly not the best methods. The use of a project management software can bring decisive advantages for project work, if carefully chosen. Especially when several people are involved in a project or sub-project and the flow of information is even greater. In essence, every project management software should cover the following basic areas:

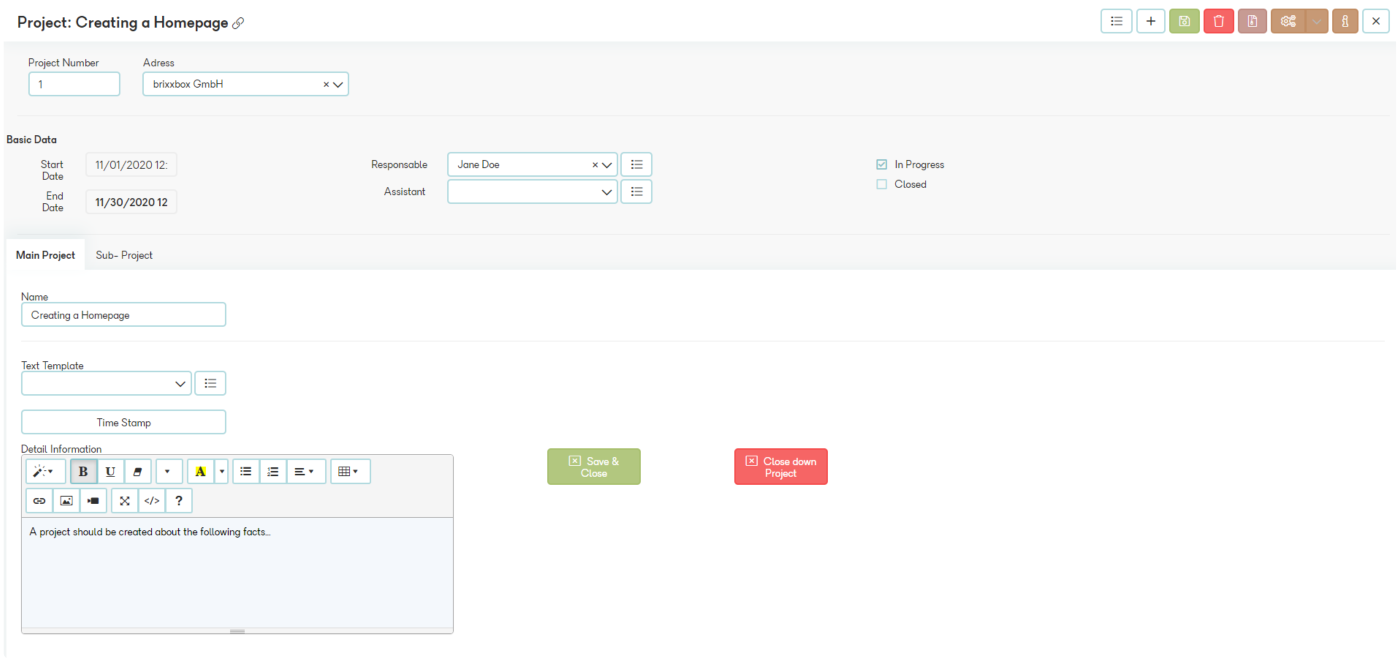
- Task management of the project
- Project planning of individual sub-projects
- Project control and resource management

If the aforementioned areas are supported by the software, it is possible for each project participant and, in addition, the management to obtain an overview of all resources and to make their own work more efficient. In the result, the advantages with the right software are obvious. It increases the possibilities of control, overview and transparency of all resources. An optimization of the personnel deployment and a higher cost control are also to be mentioned in this context.

brixxbox Low-Code-Plattform: Project Management-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.


Project: Creating a Homepage

Project Number: 1
Adress: brixxbox GmbH

Basic Data
Start Date: 11/01/2020 12:
End Date: 11/30/2020 12
Responsable: Jane Doe
Assistant:
☑ In Progress
☐ Closed

Main Project | Sub- Project

Name: Creating a Homepage
Text Template:
Time Stamp
Detail Information: A project should be created about the following facts...

In practice, this project management application could be used for example by:

- consulting company
- Freelancers
- and small enterprises

to create and manage projects. In addition to master data management, advanced functions are also included to visualize the possible use and linking of master data and to derive a realistic practical reference to support management and other departments.

Extract of the functions

- Company data: Name and address of the companies
- Team and contact persons: Contact details and contact persons of the companies and employees
- Phone book: based on contact data
- Archiving: storage and digitalization of documents and files from customers and prospects
- Templates: Text templates can be created to simplify data maintenance
- Projects: Creation and description of projects and related subprojects
- Phases: Detailed description and documentation of the projects and the respective phases
- Planning: Definition of responsibilities within projects and sub-projects

- Monitor projects
- Creation of project-relevant reminders and assignment of individual employees
- Assignment of projects to customers, or other addresses
- Resource management

You can find further information at https://brixxbox.net

Digital Visitor-Management

Contents

Who needs digital visitor management:

Companies are regularly visited by external visitors and partners. These may simply be applicants in the context of a job placement. In addition, there are of course other reasons why external visitors and partners visit a company, as the following extract makes clear:

- Machines must be maintained or repaired by technology partners
- Customers or suppliers are invited to meetings
- Audits take place

Different departments and employees are visited depending on the visitor and reason for the visit. If, for example, a machine is repaired within production, the technician inevitably gains an insight into the production processes and talks to the employees working there. It is obvious that visitors come into contact with sensitive information, some of which is important for the competitive advantages of the companies visited. Furthermore, in some industries there is even a strict documentation obligation to avoid that internal and sensitive processes are endangered by external interventions. The production of medical products is an example of this.

When recording and documenting the visits of all external persons and partners, a lot of information quickly arises, which has to be bundled, categorized and evaluated. Without a supporting technology, it does not take long until visitor management can no longer be handled by hand.

Advantages and benefits of digital visitor registration

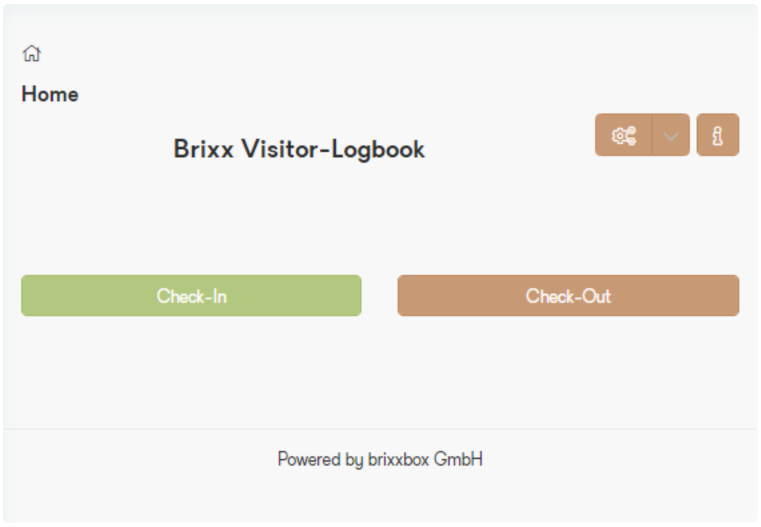
Good solutions for digital visitor management offer a variety of possibilities. Whether short or long visits, the right solution can support the entire visit process. If employees expect a visitor, relevant information can be documented in advance. With the right solution, arriving visitors have the opportunity to add relevant data directly via a touch screen. This not only makes a modern impression, but also saves a lot of time compared to documents that have to be filled in manually and then laboriously analyzed.

Applications for digital visitor management are mainly based on standard software products. Depending on the size of the company, its orientation and the number of visitors, different solutions with different functionalities can be considered. If a company grows over time, the requirements and thus the necessary functions of digital visitor management also grow. It can therefore happen that the previous functions are no longer sufficient over time and a change must be considered in order to take into account the increasing level of information for the company.

brixxbox Low-Code-Plattform: Digital Visitor-Management-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.



In practice, the digital visitor management could be implemented by:

- consulting companies
- Freelancers
- and small enterprises

to digitally manage visits and evaluate them as required.

Extract of the functions

- Create visitors (partners, customers or other guests)
- Create accompanying persons
- Define reason for visit (e.g. customer visit)
- Create visiting hours
- Evaluate visits

You can find further information at https://brixxbox.net

Data Archive

Contents

What is an archive system:

Every business process generates information that needs to be digitized and archived for later analysis and evaluation. These can be very different documents. For some time now, most companies have been electronically archiving all kinds of documents related to the transaction of goods and services such as delivery bills or invoices. Similarly, the electronic archiving of various communication channels such as email or messengers is increasing in the professional context.

An electronic archiving system offers the possibility to electronically archive documents in a certain structure. Based on the structure in the form of folders, keyworded search terms and other features, it is possible to load and visualize the previously archived data in selected form at a later time.

Depending on the type of data, additional criteria must be taken into account with regard to archiving. For example, tax-relevant digitally archived data such as invoices must be stored for 10 years in accordance with the applicable retention period in Germany and must also be audit-proof. Revision security includes the following features that must be taken into account during archiving:

- correctness of archived documents
- Completeness of archived documents
- Security of the archive procedure
- Protection against modification and falsification of archived documents
- Backup before loss of the archive system
- Use of the archive only by authorized persons
- Compliance with the retention periods
- Documentation of the archive system
- Traceability of the archive system
- Testability of the archive system

Archiving systems are mainly based on standard software products. Depending on the amount of archived data, the type of archived data and the desired integration depth, different archiving systems are possible due to the partly different requirements. If a company grows over time, the requirements and thus the necessary functions of the archiving system also grow. It can therefore happen that the previous functions of the archive system are no longer sufficient over time due to increased requirements and a change must be considered.

Advantages and benefits of an archive system

Good archiving solutions offer a wide range of possibilities and support many processes in the company. The main advantages to be mentioned in this context are

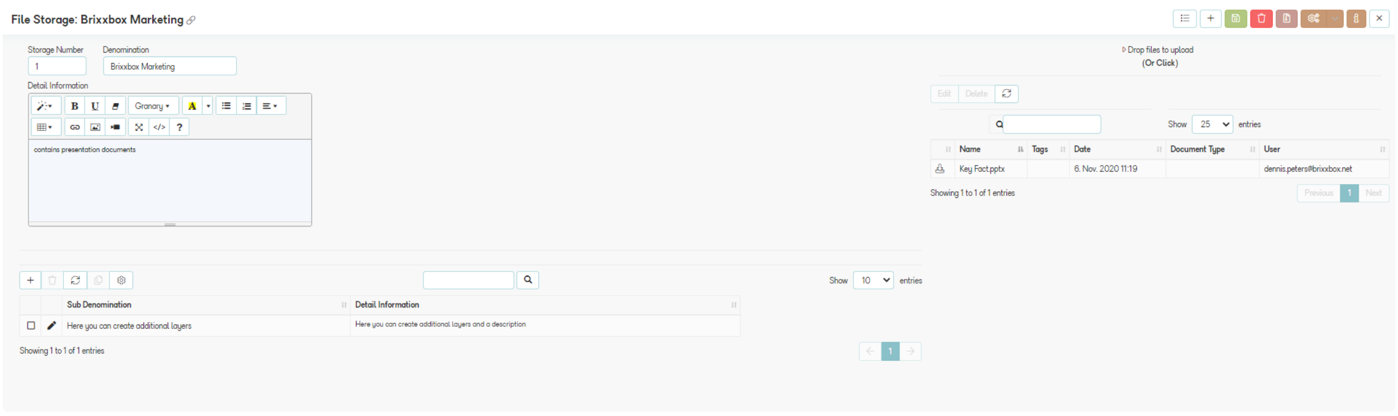
- Central data storage of archived documents and files in the electronic archive
- the reduction of search times in the electronic archive
- Saving of archive rooms with archived paper documents
- Simplified knowledge transfer through digitally archived documents
- Minimization of paper costs

Archiving systems are mainly based on standard software products. Depending on the size of the company and the type and number of documents available, different solutions with different functionalities can be considered. If a company grows over time, the requirements and thus the necessary functions of the archiving system also grow. It can therefore happen that the previous functions of the archiving system are no longer sufficient over time and a change must be considered in order to take into account the increasing level of information for the company.

brixxbox Low-Code-Plattform: Data Archive-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.



File Storage: Brixxbox Marketing

Storage Number: 1

Denomination: Brixxbox Marketing

Detail Information: contains presentation documents

		Sub Denomination	Detail Information
☐	✎	Here you can create additional layers	Here you can create additional layers and a description

Show 10 entries

Showing 1 to 1 of 1 entries

Show 25 entries

	Name	Tags	Date	Document Type	User
	Key Fact.pptx		6. Nov. 2020 11:19		dennis.peters@brixxbox.net

Showing 1 to 1 of 1 entries

In practice, the data archive template could be used for example by:

- Consulting company
- Freelancers
- and small enterprises

to manage files and documents in an orderly and clearly defined structure

Extract of the functions

- Definition of main groups and subgroups
- Archiving of any documents and files
- Description of the files
- Search function in the archive system

You can find further information at https://brixxbox.net

Ticket-System

Contents

What is a ticket system

In practice, companies regularly receive numerous inquiries. Customers and interested parties have questions about specific products and services, or need clarification regarding the products purchased. A ticket system is a software system that centrally manages all inquiries to a company. The receipt, the coordination to the right department or person and the processing of the requests is done centrally via the ticket system and bundles the channels:

- Telephone
- e-mail
- Facebook
- Instagram
- Chat and Messenger

in one system. The received requests can then be checked and assigned to a person for further processing until a solution is found (closed ticket). The ticket system is intended to ensure that no message is lost and that a complete overview of the processes to be processed is possible at any time.

Advantages and benefits of a ticket system:

Good ticket systems offer a wide range of possibilities and support many processes in the company. The main advantages to be mentioned in this context are

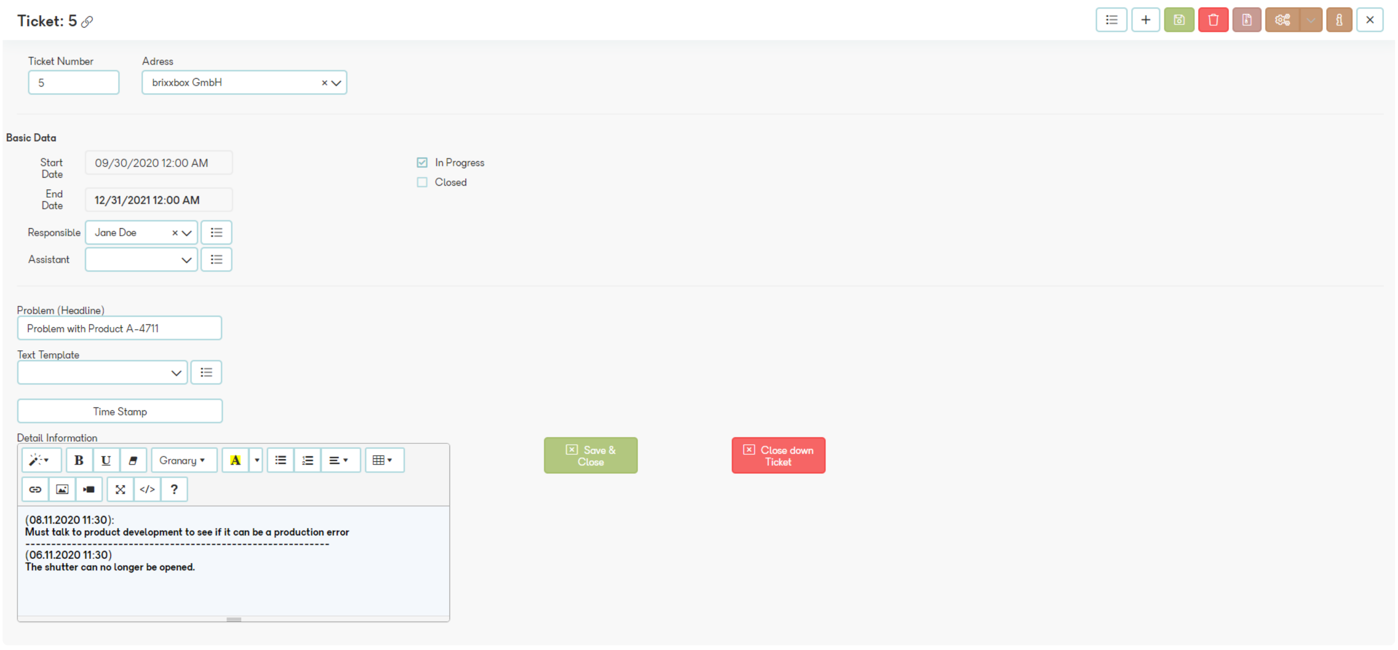
- Streamline customer inquiries: Incoming tickets are stored in a central location with a corresponding system of several communication channels. Employees can assign priorities to the tickets, monitor and process them.
- Access to customers and history: A good relationship with customers and partners is an important success factor for a company. With the ticket system, employees can also access past contacts with customers and partners, so that they always have all the facts in view
- Automation of tasks: With a One Ticket System, routine tasks can be standardized to increase efficiency and enable faster responses.
- Comprehensive evaluation options: With a good ticket system, various evaluations can be called up quickly and easily.

Ticket systems are mainly based on standard software products. Depending on the size of the company, the number and the degree of complexity of the requests, different solutions with different functionalities can be considered. If a company grows over time, the requirements and thus the necessary functions of the ticket system also grow. It can therefore happen that the previous functions of the ticket system are no longer sufficient over time and a change must be considered in order to take into account the increasing level of information for the company.

brixxbox Low-Code-Plattform: Ticket-System-Template:

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.


Ticket: 5

Ticket Number	5
Adress	brixxbox GmbH

Basic Data

Start Date	09/30/2020 12:00 AM
End Date	12/31/2021 12:00 AM
Responsible	Jane Doe
Assistant	

[✓ In Progress]
☐ Closed

Problem (Headline)	Problem with Product A-4711
Text Template	

Detail Information

(08.11.2020 11:30):
Must talk to product development to see if it can be a production error
----
(06.11.2020 11:30)
The shutter can no longer be opened.

In practice, the ticket system could be used for example by:

- consulting company
- Freelancers
- and small enterprises

to manage and document requests of all kinds.

Extract of the functions

- Company name and address of customers
- Contact person management with the corresponding contact data
- Integrated telephone book with search function
- Archiving and digitalization of documents and files (DMS)
- Entry of tickets
- Documentation of customer inquiries
- Tracking of tickets through unique numbering

You can find further information at https://brixxbox.net

Digital Adressbook

Contents

Who needs a digital address management:

Addresses, e.g. in the form of customers and suppliers, form the basis of economic activity for companies. A careful and detailed creation and administration of the respective addresses and the associated contact persons are very important. Particularly with regard to the generation and binding of new customers by suitable sales activities, many addresses accumulate fast. Without a supporting technology it does not take long until address management can no longer be handled manually.

A digital address management should at least consider the following points:

- Storage of all address relevant information
- Storage of the associated contact persons
- Storage of the respective contact information

Meanwhile, the collection of additional address information, such as interests, wishes or preferences, is also part of this. In the ideal case, a 360-degree view is achieved over time.

Advantages and benefits of digital address management

All data is in one central location. This reduces the maintenance and update effort. In practice, digital address management is in most cases linked to other systems and processes within a software from the field of "enterprise resource planning or ERP". They form the master data for relevant subsequent processes such as:

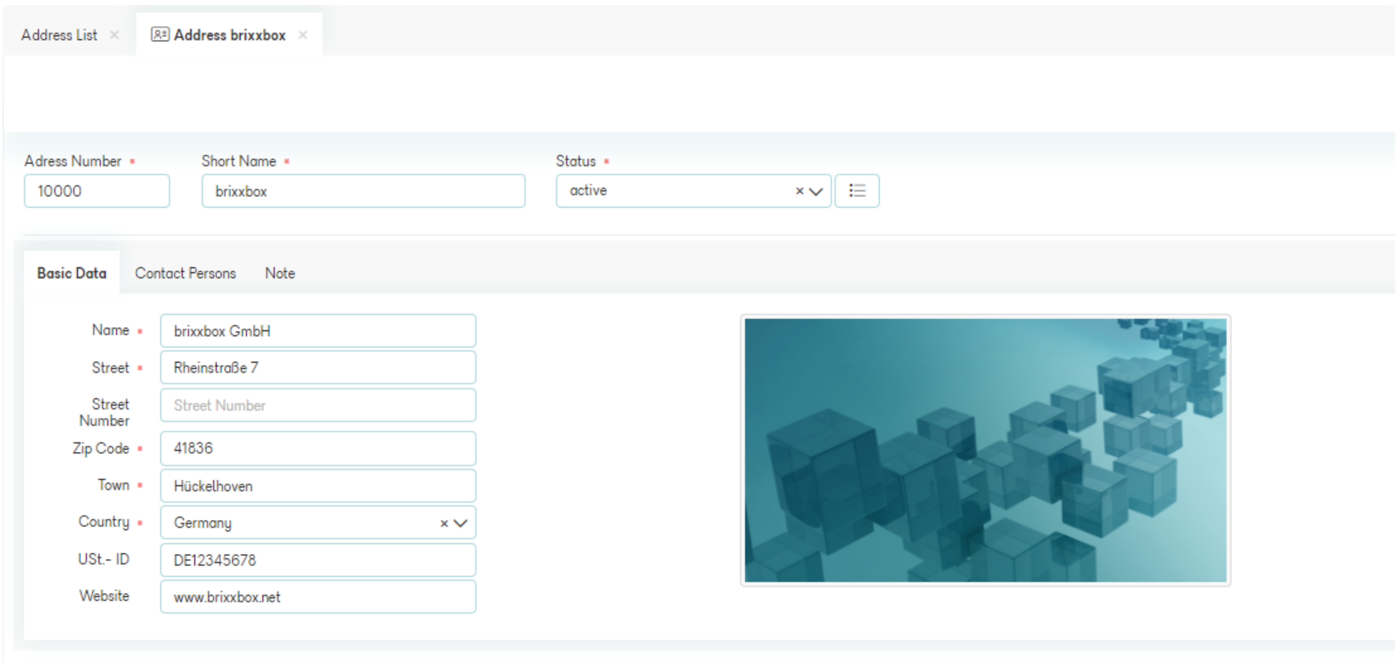
- Writing delivery bills and invoices
- or link to CRM relevant functions

Applications for digital address management in connection with relevant follow-up processes are mainly based on standard software products. Depending on the size of the company, its orientation and the desired degree of digitization, various solutions with different functionalities can be considered. If a company grows over time, the requirements and thus the necessary functions of the required solution also grow. It can therefore happen that the previous functions are no longer sufficient over time and a change must be considered in order to take into account the increasing level of information for the company.

brixxbox Low-Code-Plattform: Digital Adressbook-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.



Adress Number *	10000
Short Name *	brixxbox
Status *	active
Name *	brixxbox GmbH
Street *	Rheinstraße 7
Street Number	Street Number
Zip Code *	41836
Town *	Hückelhoven
Country *	Germany
USt.- ID	DE12345678
Website	www.brixxbox.net

In practice, the digital address management could for example be implemented by:

- consulting company
- Freelancers
- and small enterprises

to manage initial contacts and the respective contact persons after the company is founded.

Extract of the functions

- Company name and address of customers, suppliers, interested parties and other addresses
- Contact person management with the corresponding contact data
- Integrated telephone book with search function
- Archiving and digitalization of documents and files (DMS)
- Documentation of address-relevant features (e.g. specific delivery periods, acceptance deadlines, deviating conditions)

You can find further information at https://brixxbox.net

Driver's Logbook

Contents

Who needs a digital logbook:

A driver's logbook has the basic purpose of documenting the distances travelled with a vehicle as well as the respective reason for the journey. If self-employed persons or employees use a car registered with the company, the tax office generally assumes that the aforementioned groups of persons use the vehicle not only for trips of the company but also for private matters. In most cases, the pecuniary advantage resulting from the car must be taxed.

Two options are available for this purpose: Either the car is estimated with a lump sum according to the one percent rule (in Germany) - or the frequency and duration of all business and private trips with the car is documented in a detailed logbook. Although the latter method involves some effort, it can result in attractive tax benefits. However, the requirements and the declared information of the tax authorities must be met:

- A separate logbook must be kept for each vehicle
- Information on the registration number of the vehicle concerned
- Indication of the respective driver
- Information on the exact mileage at the beginning of the trip and the exact mileage at the end of the trip, as well as the driven kilometers
- Documentation of the period of the trip

The manual entry and specification of the individual trips quickly leads to a lot of work. Without a supporting technology, it does not take long until the administration is not efficient with manual work.

Advantages and benefits of a digital logbook

The use of an electronic logbook offers many advantages and often leads to lower costs, especially when there are many trips for the company and few private trips. In general, the following advantages can be mentioned:

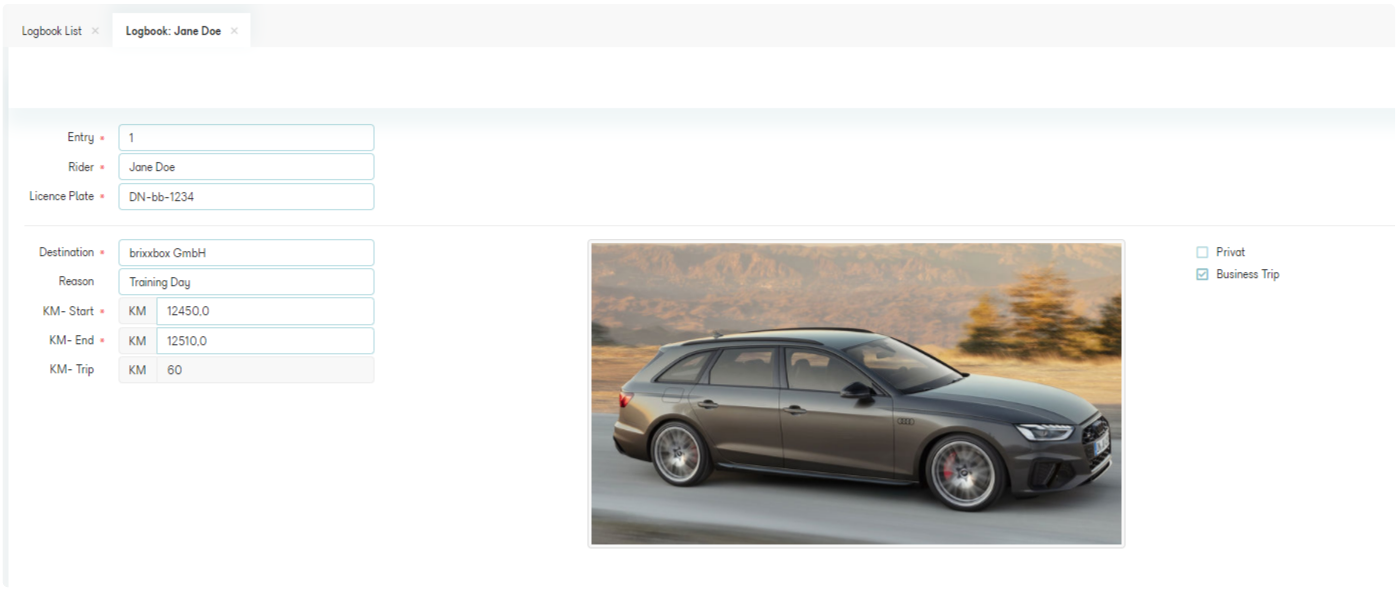
- significant time saving,
- Clear overview and hardly any errors
- Security towards the tax office,
- Generally lower costs and tax advantages compared to the one percent rule,
- electronic evaluation of the driving data.

Applications for managing digital driver's logbooks are mainly based on standard software products. Depending on the size of the company, the number of company vehicles and the degree of internationalization, different solutions with different functionalities are possible. If a company grows over time, the requirements and thus the necessary functions of the digital logbook also grow. It can therefore happen that the previous functions are no longer sufficient over time and a change must be considered in order to take into account the increasing level of information for the company.

brixxbox Low-Code-Plattform: Driver's Logbook template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.


Logbook List × Logbook: Jane Doe ×

Entry *	1
Rider *	Jane Doe
Licence Plate *	DN-bb-1234
Destination *	brixxbox GmbH
Reason	Training Day
KM- Start *	KM 12450,0
KM- End *	KM 12510,0
KM- Trip	KM 60

☐ Privat
☑ Business Trip

In practice, the digital driver's logbook could be used by:

- freelancers

to digitally manage the existing company vehicle, document the respective trips with all the data and evaluate them if required.

Extract of the functions

- Documentation of the driver, the registration number
- Recording of the kilometers at the start and at the end of the trip and determination of the driven kilometers
- Differentiation between business and private trips with regard to taxation
- Use as cell phone app possible
- Use as app for tablets possible

You can find further information at https://brixxbox.net

Order Management

Contents

Who needs digital order management:

A business model was developed and the market was explored. In the next step, products and services were declared and first addresses for acquisitions were procured. Ideally, the first customers as well as the first customer orders will be generated subsequently. The business processes are increasing. Without a supporting technology it does not take long until the workload can no longer be handled by hand.

A software for order management is a sensible solution, as long as it takes at least the following points into account:

- Storage of all address-relevant information including contact persons and contact information in the software
- Storage of existing articles and services in the software
- Creation of customer orders and status management in the software

Meanwhile the collection of additional information, like special places of delivery, certain delivery times or other preferences counts likewise to it.

Advantages and benefits of digital order management

In such a system, all data is located in one central location. Addresses, articles and other master data can be effectively managed in the software and updated as required. In practice, order management is in most cases linked to other systems and processes within a software from the field of "enterprise resource planning or ERP". These include, for example

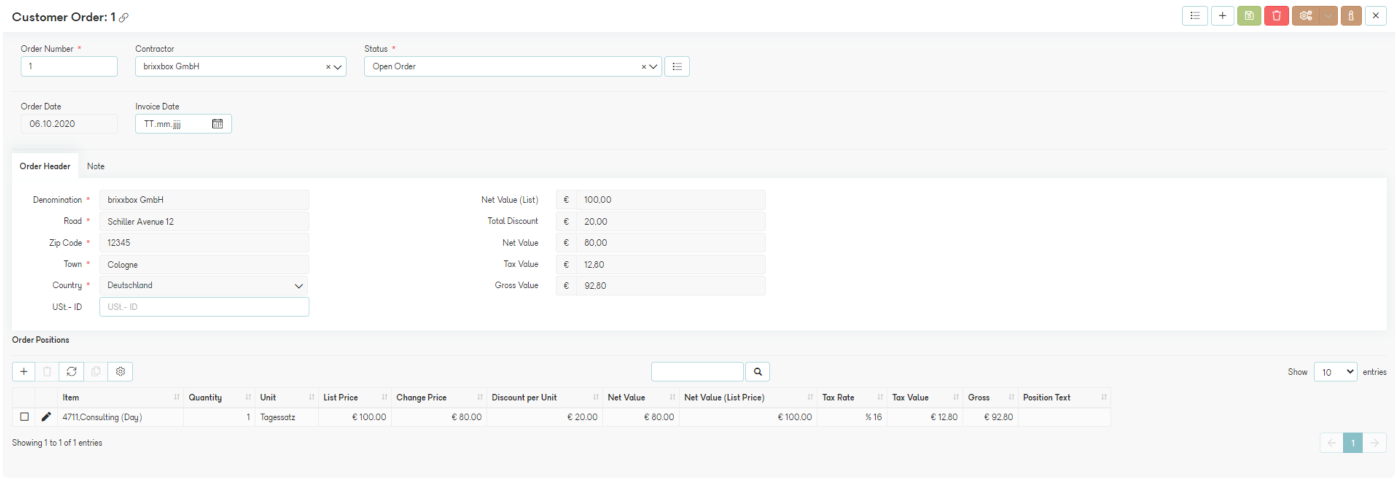
- Billing programs
- CRM systems to directly address customer needs
- Archiving system for archiving of documents

Applications for digital job management in connection with relevant follow-up processes are mainly based on standard software products. Depending on the size of the company, its orientation and the desired level of digitization, various solutions with different functionalities are possible. If a company grows over time, the requirements and thus the necessary functions of the required solution also grow. It can therefore happen that the previous functions are no longer sufficient over time and a change must be considered in order to take into account the increasing level of information for the company.

brixxbox Low-Code-Plattform: Digital Order Management-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.


Customer Order: 1

Order Number *	1
Contractor	brixxbox GmbH
Status *	Open Order
Order Date	06.10.2020
Invoice Date	TT.mm.jjjj

Order Header | Note

Denomination *	brixxbox GmbH	Net Value (List)	€ 100,00
Road *	Schiller Avenue 12	Total Discount	€ 20,00
Zip Code *	12345	Net Value	€ 80,00
Town *	Cologne	Tax Value	€ 12,80
Country *	Deutschland	Gross Value	€ 92,80
USt.- ID	USt.- ID		

Order Positions

Show 10 entries

Item	Quantity	Unit	List Price	Change Price	Discount per Unit	Net Value	Net Value (List Price)	Tax Rate	Tax Value	Gross	Position Text
4711,Consulting (Day)	1	Tagessatz	€ 100.00	€ 80.00	€ 20.00	€ 80.00	€ 100.00	% 16	€ 12.80	€ 92.80	

Showing 1 to 1 of 1 entries

In practice, the digital order management could be used for example by:

- consulting company
- Freelancers
- and small enterprises

can be used to manage initial contacts with the respective contact persons after the company is founded and to monitor initial business activities (e.g. customer orders).

Extract of the functions

- Company name and address of customers, suppliers, interested parties and other addresses
- Contact person management with the corresponding contact data
- Archiving and digitalization of documents and files (DMS)
- Documentation of address-relevant features (e.g. specific delivery periods, acceptance deadlines, deviating conditions, etc.)
- Article management
- Price management
- Entry and control of discounts per order
- Enter and document order
- Manage status of the job

You can find further information at https://brixxbox.net

Basic CRM

Contents

What is CRM

Customer Relationship Management (CRM) is a strategy for companies and their employees to systematically create relationships and interactions with contact persons of existing and potential customers. By generating sales, these customers play a decisive role in the long-term success of the company. If customers adjust their buying behavior and switch to products from other companies, this can have serious consequences for their own company. That's why companies invest a lot of time and effort in CRM activities to analyze customer preferences. Target groups, markets and the respective needs are analyzed in the CRM and coordinated with the own portfolio. If deviations are found, the employees of the sales and marketing department, in coordination with the management, have the task of optimizing the customer relationship within the framework of CRM or adjusting the concrete CRM strategies in order to better reach the customers again. Other departments also contribute to customer satisfaction and provide valuable information for CRM. This includes, for example, the entire service department. Within the scope of service, customer inquiries and discrepancies are recorded and processed. This is also part of CRM.

CRM generates a wide range of detailed information that must be bundled, categorized and evaluated in order to derive statements. Without the appropriate technology, with increasing numbers of customers and prospects and the respective contacts, it doesn't take long before CRM can no longer be managed with manual work. Targeted contact with the customer is made more difficult. Without a suitable technology, employees lose too much time for CRM if the CRM strategy is to be pursued with the same quality and intensity even with increasing numbers of customers and prospects.

What is a CRM system

A CRM system is a software with functions for the management and automation of all information that is generated month by month when addressing customers or prospects and binding them. In the CRM software, all data of customers and prospects, including all transactions and related communication, are collected and stored in databases. The following information can be taken as an example:

- Documentation of a prospective customer telephone call for later evaluation
- Storage of a customer offer for later allocation
- Appointment reminder for an offer meeting with an interested party on site
- Birthday management of A and B customers to increase attention

CRM systems are mainly based on standard software products. Depending on the size of the company and the required level of information, different solutions with different functionalities are possible. If a company grows over time, the requirements and thus the necessary functions of the CRM system also grow. It can therefore happen that the previous functions of the CRM software are no longer sufficient over time and a change must be considered in order to take into account the increasing degree of information for the company.

Advantages and benefits of a CRM system

The advantages and benefits of CRM software for companies are obvious. In practice, a large number of contacts and their relationship to your own company must be managed in one system. Well-designed CRM systems create more transparency. They can be used wherever direct or indirect customer or prospect contact is required. With the right CRM software, the entire sales force can also be supported. This applies both to:

- the sales field service with the objective of "increasing sales",
- as well as for the service-oriented field service with the objective "service and maintenance".

All on-site activities can be recorded directly in the CRM software. In addition, a good CRM system can also be an aid for certain activities, in which information is available at the right place in the right form. This not only saves time, but also regularly delivers better results in all areas.

In summary, a CRM system helps to support sales, marketing, service and last but not least, management. With the help of a CRM system, activities can be optimized and automated (e.g. monthly reports). In addition, a good CRM provides a variety of decision bases.

brixxbox Low-Code-Plattform: Basic CRM-Template

With the "brixxbox" cloud software, companies can digitize processes quickly and easily. Thanks to the underlying modular principle, necessary adjustments and extensions can be made in the software at any time at the request of management or other employees in order to process additional data in the system. Adjustments in the system do not affect the productive operation of the software in the respective company in any way.

The provided templates serve as configuration examples. Different approaches shall be illustrated. The templates can be supplemented after the import as desired, or used as templates for own applications or entire systems.

In practice, this CRM application could be used for example by:

- freelancers
- and small enterprises

to create customers and prospects and manage their contacts and CRM-relevant data. In addition to master data management, advanced functions are also included to visualize the possible use and linking of master data and to derive a realistic practical reference to support management, sales and marketing.

Extract of the functions

- Company data: Name and address of the companies
- contact details: Contact data and contact persons of the companies and employees
- Phone book: based on contact data
- Archiving: storage and digitalization of documents and files from customers and prospects
- CRM data: Scheduling (e.g. per month, week), deadline monitoring and appointment documentation as well as assignment of relevant contacts
- CRM Workflows: Creation of reminders and follow-ups (e.g. in two months) for certain contacts
- Templates: Text templates can be created
- Mobile: Access to all data anywhere and with any device (cloud)

You can find further information at https://brixxbox.net

Erweitertes CRM-System

Contents

Was ist CRM

Das Customer Relationship Management (kurz CRM) ist für Unternehmen und deren Mitarbeiter eine Strategie zur systematischen Gestaltung von Beziehungen und Interaktionen mit Kontaktpersonen bestehender und potenzieller Kunden. Diese sind durch die Umsatzgenerierung maßgeblich am langfristigen Erfolg des Unternehmens beteiligt. Wenn Kunden Ihr Kaufverhalten anpassen und auf Produkte anderer Unternehmen ausweichen, kann dies für das eigene Unternehmen ernste Folgen haben. Daher investieren Unternehmen viel Zeit und Aufwand in die CRM-Tätigkeiten zur Analyse der Kundenwünsche. Zielgruppen, Märkte und die jeweiligen Bedürfnisse werden im Rahmen des CRM analysiert und mit dem eigenen Portfolio abgestimmt. Werden Abweichungen festgestellt, obliegt den Mitarbeitern des Vertriebs und des Marketings in Abstimmung mit dem Management die Aufgabe das Kundenverhältnis im Rahmen des CRM zu Optimieren bzw. die konkreten CRM-Strategien anzupassen, um die Kunden wieder besser zu erreichen. Auch andere Abteilungen tragen zur Kundenzufriedenheit bei und liefern wertvolle Informationen für das CRM. Hierzu zählt zum Beispiel der gesamte Bereich Service. Im Rahmen des Service werden Kundenanfragen und Unstimmigkeiten erfasst und bearbeitet. Auch dies ist Teil des CRM.

Beim CRM entstehen vielfältige, detaillierte Informationen, die gebündelt, kategorisiert und bewertet werden müssen, um Aussagen ableiten zu können. Ohne eine entsprechende Technologie dauert es bei steigenden Kunden-, und Interessentenzahlen und den jeweiligen Kontakten nicht lange, bis CRM mit händischer Arbeit nicht mehr zu bewältigen ist. Der zielgerichtete Kontakt zum Kunden wird erschwert. Die Mitarbeiter verlieren ohne eine geeignete Technologie zu viel Zeit für das CRM, wenn die CRM-Strategie auch bei steigenden Kunden und Interessenten mit gleicher Qualität und Intensität verfolgt werden soll.

Was ist ein CRM-System

Ein CRM-System ist eine Software mit Funktionen zur Verwaltung und Automatisierung aller Informationen, die bei der Kunden-, bzw. Interessentenansprache und deren Bindung Monat für Monat anfallen. In der CRM-Software werden sämtliche Daten von Kunden und Interessenten inklusive aller Transaktionen und der damit verbundenen Kommunikation erfasst und in Datenbanken gespeichert. Dies können beispielhaft folgende Informationen sein:

- Dokumentation eines Interessententelefonates zur späteren Bewertung
- Ablage eines Kundenangebotes zur späteren Zuordnung
- Terminerinnerung für eine Angebotsbesprechung bei einem Interessenten vor Ort
- Geburtstagsverwaltung der A- und B-Kunden zur Aufmerksamkeitssteigerung

CRM-Systeme basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße und dem notwendigen Informationsgrad kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des CRM-System. Es kann daher vorkommen, dass die bisherigen Funktionen der CRM-Software mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

Vorteile und Nutzen eines CRM-Systems

Die Vorteile und der Nutzen einer CRM-Software für Unternehmen liegen klar auf der Hand. In der Praxis muss eine Vielzahl von Kontakten und deren Beziehung zum eigenen Unternehmen in einem System verwaltet werden. Gut durchdachte CRM-Systeme schaffen mehr Transparenz. Sie können überall dort eingesetzt werden, wo direkter oder indirekter Kunden-, bzw. Interessentenkontakt erforderlich ist. Mit der richtigen CRM-Software kann ebenfalls der gesamte Außendienst unterstützt werden. Dies gilt sowohl für:

- den vertrieblichen Außendienst mit der Zielsetzung "Umsatzsteigerung",
- als auch für den serviceorientierten Außendienst mit der Zielsetzung "Service und Wartung".

Alle Aktivitäten vor Ort können direkt in die CRM-Software erfasst werden. Darüber hinaus kann ein gutes CRM-System auch eine Hilfestellung für bestimmte Tätigkeiten sein, in dem Informationen an der richtigen Stelle in der richtigen Form aufbereitet zur Verfügung stehen. Dies spart nicht nur Zeit, sondern liefert regelmäßig auch bessere Resultate in allen Bereichen.

Zusammengefasst trägt ein CRM-System dazu bei den Vertrieb, das Marketing, den Service und nicht zuletzt das Management zu unterstützen. Mit Hilfe eines CRM-Systems können Tätigkeiten optimiert und automatisiert (z.B. monatliche Reports) werden. Darüber hinaus liefert ein gutes CRM vielfältige Entscheidungsgrundlagen.

Vorteile und Nutzen eines integrierten Projektmanagementsystems

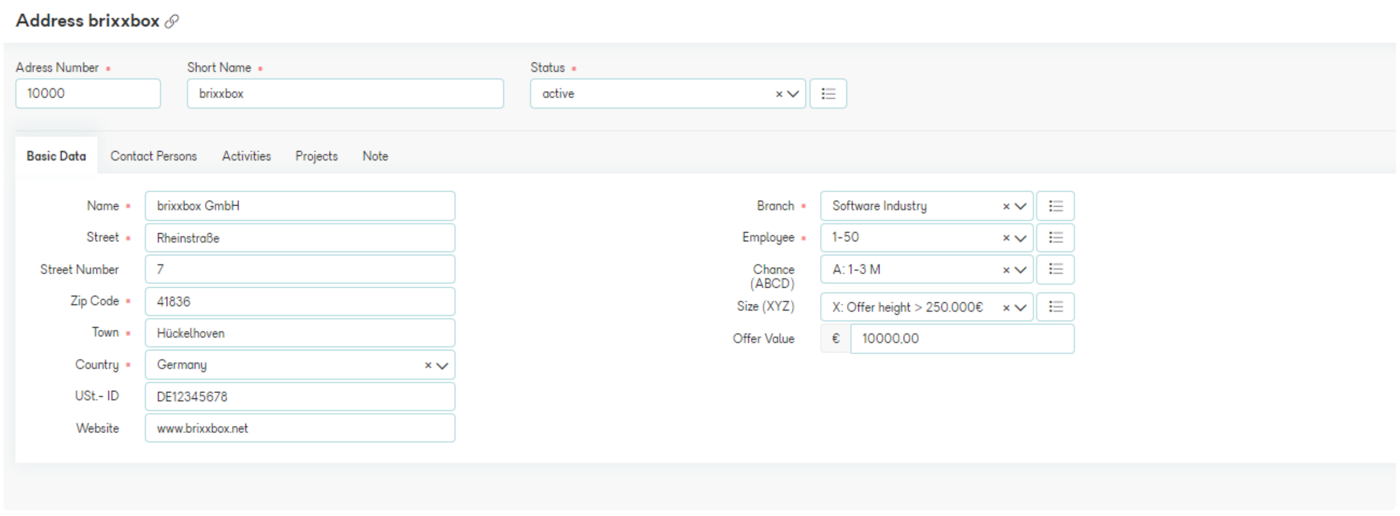
Projekte sind regelmäßig mit Investitionen und sonstigen Aufwänden verbunden und auf ein bestimmtes Ziel (Projektziel) gerichtet. In der Praxis wird ein Projekt häufig in Teilprojekte gelgliedert, die jeweils auf eine bestimmte (Teil-) Zielerfüllung gerichtet sind und mit dem Hauptziel in Verbindung stehen. Dem Projektmanagement obliegt hierbei die Aufgabe die (Teil-) Projekte zu planen und zu steuern, sowie einer Überwachung im Hinblick auf zeitlich-, und ressourcenorientierte Parameter. Wie auch beim CRM handelt es sich beim Projektmanagement um Strategien, die spätestens ab einem bestimmten Komplexitätslevel durch geeignete Technologien und Systeme unterstützt werden müssen, um die Fülle an Daten sinnvoll zu verarbeiten. Ein CRM-System mit integriertem Projektmanagement spart Zeit und Aufwand und erhöht die Datenqualität.

Der Vertrieb akquiriert beispielhaft ein neues Kundenprojekt. Bisher wurden alle kunden-, bzw. interessentenseitigen Aktivitäten im CRM-System erfasst. Das nun folgende Projekt wird regelmäßig von anderen Mitarbeitern und Abteilungen gemanagt. Durch die Integration finden Sie nun auch projektabhängige Informationen in kürzester Zeit. Das Management der Informationen wird durch die Symbiose beider Systeme deutlich optimiert.

brixxbox Low-Code-Plattform: Erweitertes CRM-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.


Address brixxbox

Adress Number * 10000
Short Name * brixxbox
Status * active

Basic Data | Contact Persons | Activities | Projects | Note

Name *	brixxbox GmbH	Branch *	Software Industry
Street *	Rheinstraße	Employee *	1-50
Street Number	7	Chance (ABCD)	A: 1-3 M
Zip Code *	41836	Size (XYZ)	X: Offer height > 250.000€
Town *	Hückelhoven	Offer Value	€ 10000,00
Country *	Germany		
USt.- ID	DE12345678		
Website	www.brixxbox.net		

In der Praxis könnte diese CRM-Anwendung zum Beispiel von:

- Beratungsunternehmen
- Freelancern
- und Kleinunternehmen

genutzt werden, um Kunden und Interessenten anzulegen und deren Kontakte und CRM-relevante Daten zu verwalten. Neben der Stammdatenhaltung sind darüber hinaus erweiterte Funktionen enthalten, um eine mögliche Verwendung und Verknüpfung der Stammdaten zu visualisieren und einen realistischen Praxisbezug zur Unterstützung des Managements, des Vertriebs und des Marketings herzuleiten.

Auszug der Funktionen

- Unternehmensdaten: Bezeichnung und Anschrift der Unternehmen
- Kontaktdaten: Kontaktdaten und Ansprechpartner der Unternehmen und Mitarbeiter
- Telefonbuch: auf Basis der Kontaktdaten
- Archivierung: Ablage und Digitalisierung von Dokumenten und Dateien von Kunden und Interessenten
- Besonderheiten: Dokumentation von spezifischen Kundenwünschen und Besonderheiten (z.B. Abrechnung nur pro Monat)
- CRM Daten: Terminplanung (z.B. pro Monat, Woche), Terminüberwachung und Termindokumentation sowie Zuordnung relevanter Kontakte
- CRM Workflows: Erstellung von Erinnerungen und Wiedervorlagen (z.B. in zwei Monaten) für bestimme Kontakte
- CRM Reports: z.B. ABC/XYZ - Bewertung oder Kategorisierung von Kunden und Interessenten (Perioden/Monate...)
- Vorlagen: Textvorlagen können erstellt werden
- Projektintegration: Projektplanung, Projektüberwachung, Ressourcenverwaltung und Zuständigkeiten
- Management: Statusüberwachung aller Aktivitäten und Projekte
- Mobil: Zugriff auf alle Daten überall und mit jedem Endgerät möglich (Cloud)

Weitere Informationen finden Sie unter https://brixxbox.net

Zeiterfassung

Contents

Warum benötige ich eine Zeiterfasung:

In vielen Unternehmen wurde für einige Mitarbeiter lange Zeit das Prinzip der Vertrauensarbeitszeit praktiziert. Hierbei handelt es sich um ein Organisationsmodell, bei dem die Erledigung der Aufgaben im Vordergrund steht. Nicht die Arbeitszeit, sondern das Arbeitsergebnis ist entscheidend. Dieses Modell wurde vor allem von kreativ arbeitenden Mitarbeitern praktiziert, da eine starre Arbeitszeitvorgabe auch aus wissenschaftlichen Gesichtspunkten die Kreativität und damit vor allem mögliche Innovationen behindert.

Der europäische Gerichtshof hat jedoch im Jahr 2019 entschieden, dass die Erfassung von Arbeitszeiten künftig in allen Unternehmen von allen Mitarbeitern zu erfolgen hat. Durch das Urteil werden Arbeitsgeber dazu verpflichtet, ein Konzept zur Zeiterfassung anzubieten, durch welches die gesamte Arbeitszeit sowie Pausen und Fehltage durch Urlaub oder Krankheut der Mitarbeiter dokumentiert werden können. So soll sichergestellt sein, dass die Gesetze im Hinblick auf die Arbeits-, und Pausenzeiten sowie die gesetzlich vorgeschrieben Urlaubstage zum Schutz des Mitarbeiters eingehalten werden.

Bei der Erfassung und Dokumentation der Arbeitszeiten aller Mitarbeiter entstehen schnell viele Informationen, die gebündelt, kategorisiert und bewertet werden müssen, um die Einhaltung der vorgeschriebenen Gesetze zu gewährleisten. Ohne eine unterstützende Technologie dauert es nicht lange, bis die Zeiterfassung mit händischer Arbeit nicht mehr zu bewältigen ist.

Was ist ein Zeiterfassungssystem:

Ein Zeiterfassungssystem ist eine Software mit Funktionen zur Dokumentation und Auswertung aller zeitorientierten Parameter der jeweiligen Mitarbeiter, die der Gesetzgeber vorsieht. In der Zeiterfassungssoftware werden alle relevanten Daten der Mitarbeiter in Datenbanken gespeichert. Hierzu zählen insbesondere die folgenden Bereiche:

- Arbeitszeiten
- Pausenzeiten
- Urlaubstage
- Fehltage durch Krankheit

Zeiterfassungssysteme basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße und den verschiedenen Arbeitszeitmodellen wie z.B. Schichtarbeit oder Akkordarbeit kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des Zeiterfassungssystems. Es kann daher vorkommen, dass die bisherigen Funktionen der Zeiterfassungssoftware mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

Vorteile und Nutzen eines Zeitefassungssystems

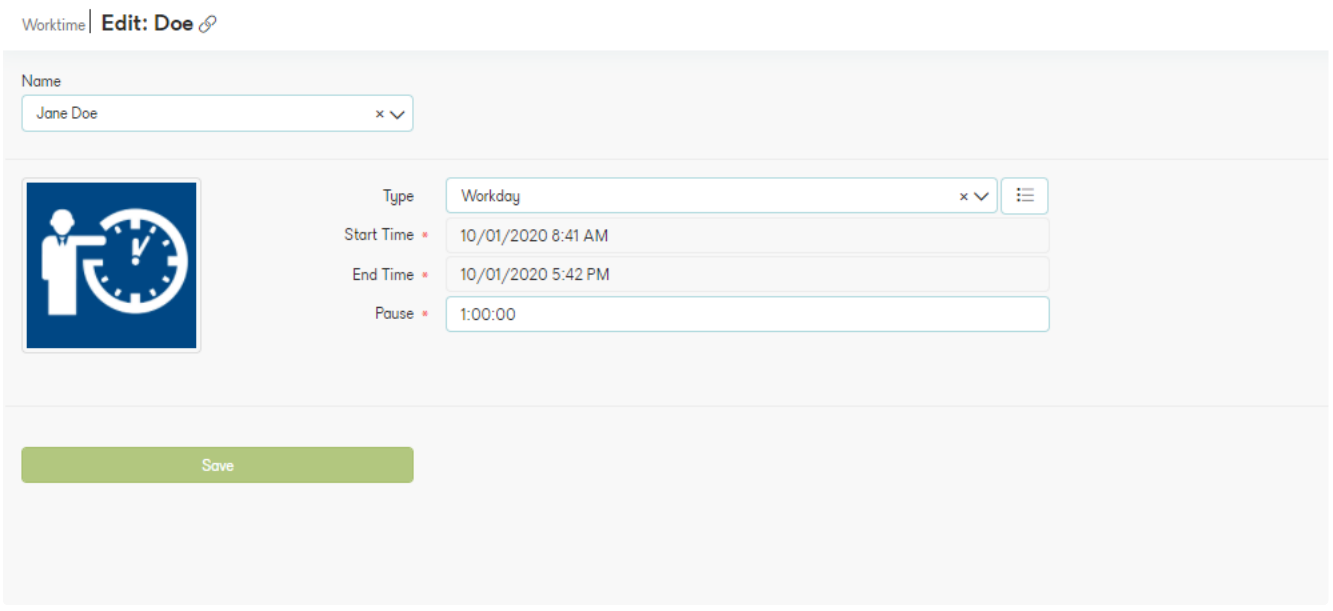
Gute Zeiterfassungssysteme bieten eine enorme Flexibilität im Hinblick auf die Dokumentation der betrieblichen Arbeitszeit der Belegschaft. Es gibt eine Vielzahl von Erfassungsmöglichkeiten, die je nach eingesetzter Software möglich sind. So gibt es beispielsweise Systeme, die eine Eingabe mittels Mobiltelefon möglich machen. Dies ist vor allem für den Außendienst interessant. So können die Daten schnell und ohne viel Aufwand in das System eingegeben werden. Natürlich gibt auch andere Endgeräte, die abhängig vom System kompatibel sind. Hierzu zählen beispielsweise Terminals mit Chip-Karten oder vergleichbare Systeme, die es dem Arbeitgeber durch einfaches auflegen ermöglichen die jeweiligen Zeittypen zu buchen.

Darüber hinaus kann ein gutes Zeiterfassungssystem auch eine Hilfestellung für bestimmte Entscheidungen des Managements oder der Personalabteilung sein und nicht zuletzt liefert ein Zeiterfassungssystem regelmäßig die Basis für die Engeltabrechnung.

brixxbox Low-Code-Plattform: Zeiterfassungs-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.



Edit: Doe

Name	Jane Doe
Type	Workday
Start Time *	10/01/2020 8:41 AM
End Time *	10/01/2020 5:42 PM
Pause *	1:00:00

In der Praxis könnte diese Zeiterfassungs-Anwendung zum Beispiel von:

- Freelancern
- und Kleinunternehmen

genutzt werden, um die eigenen Arbeitszeiten zu dokumentieren, sowie den Jahresurlaub zu planen.

Auszug der Funktionen

- Mitarbeiter: Mitarbeiterverwaltung mit den zugehörigen Kontaktdaten
- Erfassung: Erfassung der täglichen Arbeitszeiten und Pausen der jeweiligen Mitarbeiter
- Planung: Dokumentation von Urlaubsanträgen und Statusverwaltung der jeweiligen Mitarbeiter
- Status: Dokumentation von Genehmigungen

Weitere Informationen finden Sie unter https://brixxbox.net

Digitales Besuchermanagement

Contents

Wer benötigt ein digitales Besuchermanagement:

Regelmäßig werden Unternehmen von externen Personen und Partnern besucht. Dies können schlicht Bewerber im Rahmen einer Stellenbesetzung sein. Darüber hinaus gibt es natürlich weitere Gründe, warum externe Personen und Partner einen Betrieb besuchen, wie der nachfolgende Auszug deutlich macht:

- Maschinen müssen durch Technologiepartner gewartet oder repariert werden
- Kunden oder Lieferanten werden zu Besprechungen eingeladen
- Es finden Audits statt

Je nach Besucher und Besuchsgrund werden unterschiedliche Abteilungen und Mitarbeiter besucht. Wird beispielsweise eine Maschine innerhalb der Produktion repariert, erlangt der Monteur zwangläufig einen Einblick in die Produktionsprozesse und spricht mit den dort arbeitenden Mitarbeitern. Es liegt auf der Hand, dass Besucher in Kontakt mit sensiblen Informationen kommen, die zum Teil wichtig für die Wettbewerbsvorteile der besuchten Unternehmen sind. Darüber hinaus besteht in einigen Branchen sogar eine strenge Dokumentationspflicht, um zu vermeiden das interne und sensible Prozesse durch externe Eingriffe gefährdet werden. Exemplarisch zu nennen ist hier die Produktion von Medizinprodukten.

Bei der Erfassung und Dokumentation der Besuche aller externen Personen und Partner entstehen schnell viele Informationen, die gebündelt, kategorisiert und ausgewertet werden müssen. Ohne eine unterstützende Technologie dauert es nicht lange, bis die Besucherverwaltung mit händischer Arbeit nicht mehr zu bewältigen ist.

Vorteile und Nutzen einer digitalen Besuchererfassung

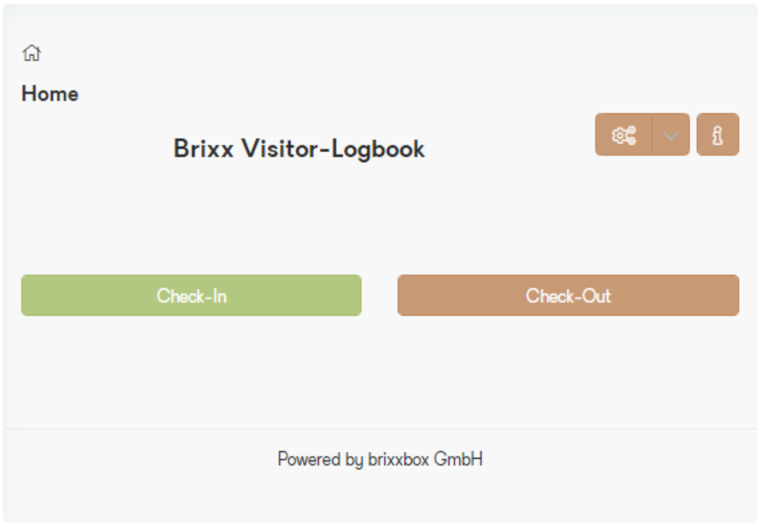
Gute Lösungen zur digitalen Besucherverwaltung bieten vielfältige Möglichkeiten. Ob kurzer oder langer Besuch, mit der richtigen Lösung kann der gesamte Besuchsprozess unterstützt werden. Wenn Mitarbeiter einen Besucher erwarten, können bereits im Vorfeld relevante Informationen dokumentiert werden. Ankommende Besucher haben mit dem richtigen Lösung die Möglichkeit entsprechende Daten direkt über einen Touch-Bildschirm zu ergänzen. Das macht nicht nur einen modernen Eindruck, sondern spart darüber hinaus auch viel Zeit im Vergleich zu manuell auszufüllenden Dokumenten, die in der Folge mühsam analysiert werden müssen.

Anwendungen zur digitalen Besucherverwaltung basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße, der Ausrichtung und dem Besucheraufkommen kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen der digitalen Besucherverwaltung. Es kann daher vorkommen, dass die bisherigen Funktionen mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

brixxbox Low-Code-Plattform: digitale Besucherverwaltung-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.



In der Praxis könnte das digitale Besuchermanagement zum Beispiel von:

- Beratungsuntermehmen
- Freelancern
- und Kleinunternehmen

genutzt werden, um die Besuche digital zu verwalten und bei Bedarf auszuwerten.

Auszug der Funktionen

- Besucher anlegen (Partner, Kunden oder weitere Gäste)
- Begleitpersonen anlegen
- Besuchsgrund definieren (z.B. Kundenbesuch)
- Besuchszeiten anlegen
- Besuche auswerten

Weitere Informationen finden Sie unter https://brixxbox.net

Datenarchiv

Contents

Was ist ein Archivsystem:

Bei jedem Geschäftsprozess entstehen Informationen, die für die spätere Auswertbarkeit und Analyse digitalisiert und archiviert werden müssen. Dies können ganz unterschiedliche Dokumente sein. So werden seit geraumer Zeit bereits in den meisten Unternehmen jeglichen Dokumente im Zusammenhang mit der Transaktion von Waren und Dienstleitungen wie Lieferscheine oder Rechnungen elektronisch archiviert. Ebenso nimmt die elektronische Archivierung verschiedener Kommunikationswege wie Email oder Messenger im beruflichen Kontext zu.

Ein elektronisches Archivsystem bietet die Möglichkeit Dokumente in einer bestimmten Struktur elektronisch zu archivieren. Anhand der Struktur in Form von Ordnern, verschlagworteten Suchbegriffen und anderen Merkmalen ist es zu einem späteren Zeitpunkt möglich die zuvor archivierten Daten in selektierter Form zu laden und zu visualisieren.

Abhängig von der Datenart müssen im Hinblick auf die Archivierung zusätzliche Kriterien berücksichtigt werden. So müssen beispielsweise steuerrelevante digital archivierte Daten wie Rechnungen gemäß der geltenden Aufbewahrungsfrist in Deutschland 10 Jahre gespeichert werden und zusätzlich Revisionssicher sein. Revisionssicherheit umfasst folgende Merkmale, die bei der Archivierung berücksichtigt werden müssen:

- Richtigkeit der archivierten Dokumente
- Vollständigkeit der archivierten Dokumente
- Sicherheit des Archiv-Verfahrens
- Schutz vor Veränderung und Verfälschung der archivierten Dokumente
- Sicherung vor Verlust des Archivsystems
- Nutzung des Archivs nur durch Berechtigte
- Einhaltung der Aufbewahrungsfristen
- Dokumentation des Archivsystems
- Nachvollziehbarkeit des Archivsystems
- Prüfbarkeit des Archivsystems

Archivsysteme basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der archivierten Datenmenge, der archivierten Datenart und der gewünschten Integrationstiefe kommen verschiedene Archivsysteme aufgrund der teils unterschiedlichen Anforderungen in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des Archivsystems. Es kann daher vorkommen, dass die bisherigen Funktionen des Archivsystems mit der Zeit aufgrund gestiegener Anforderungen nicht mehr ausreichen und ein Wechsel erwogen werden muss.

Vorteile und Nutzen eines Archivsystems

Gute Archivierungslösungen bieten vielfältige Möglichkeiten und unterstützen viele Prozesse im Unternehmen. Die in diesem Zusammenhang zu nennenden Hauptvorteile sind:

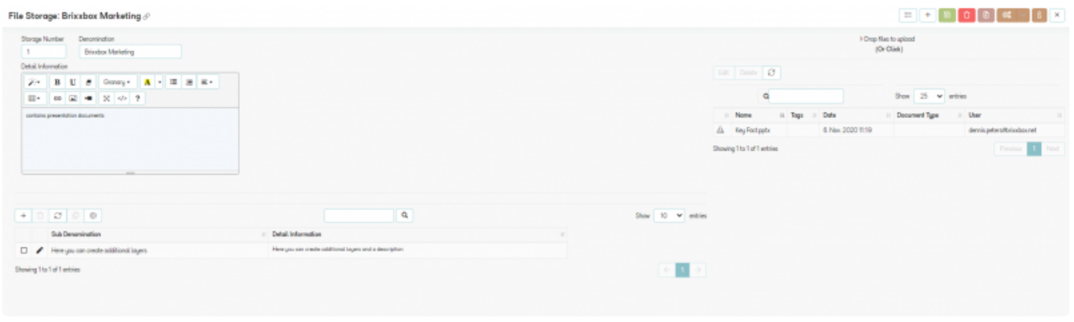
- die Zentrale Datenspeicherung von archivierten Dokumenten und Dateien im elektronischen Archiv
- die Reduktion von Suchzeiten im elektronischen Archiv
- Einsparung von Archivräumen mit archivierten Papierdokumenten
- Vereinfachter Wissenstranfer durch digital archivierte Dokumente
- Minimierung der Papierkosten

Archivsysteme basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße, der Art und Anzahl der vorhandenen Dokumenten kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des Archivsystems. Es kann daher vorkommen, dass die bisherigen Funktionen des Archivsystems mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

brixxbox Low-Code-Plattform: Datenarchiv-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.


File Storage: Brixxbox Marketing

Storage Number: 1
Denomination: Brixxbox Marketing
Detail Information: contains presentation documents

Drop files to upload
(Or Click)

Show 25 entries

Name	Tags	Date	Document Type	User
Key Fact.pptx		6. Nov. 2020 11:19		dennis.peters@brixxbox.net

Showing 1 to 1 of 1 entries

Show 10 entries

Sub Denomination	Detail Information
Here you can create additional layers	Here you can create additional layers and a description

Showing 1 to 1 of 1 entries

In der Praxis könnte das Datenarchiv-Template zum Beispiel von:

- Beratungsuntermehmen
- Freelancern
- und Kleinunternehmen

genutzt werden, um Dateien und Dokumente in einer geordneten und klar definierten Struktur zu verwalten

Auszug der Funktionen

- Definition von Haupt-, und Untergruppen
- Archivierung von beliebigen Dokumenten und Dateien
- Beschreibung der Dateien
- Suchfunktion im Archivsystem

Weitere Informationen finden Sie unter https://brixxbox.net

Projektmanagement

Contents

Was ist Projektmanagement:

Das Umfeld eines Unternehmens unterliegt einem permanentem Wandel. So ändern sich in bestimmten Phasen rechtliche Rahmenparameter oder neue Technologien führen zu veränderten Kundenbedürfnissen. Auch wenn dies nur Auszüge aus einem komplexen Gesamtgefüge sind, so wirken sie sich dennoch regelmäßig auf den Erfolg eines Unternehmens aus. Es entstehen neue Aufgaben. Ursprüngliche Ziele müssen hinterfragt und von Fall zu Fall im Rahmen von Projekten und abgeleiteten Aufgaben angepasst werden. Per Definition handelt es sich bei einem Projekt um einen einmaligen (Geschäfts-)Prozess, der in verschiedene Phasen und Aufgaben aufgeteilt und auf eine bestimmte Zielerreichung gerichtet ist sowie mit einem festen Start-, und Endzeitpunkt versehen ist.

In diesem Kontext müssen Ziele definiert werden, Wege zur Zielerreichung gefunden und gemanagt werden, sowie Projektteilnehmer als Team und Ressourcen überwacht werden. Die Bündelung dieser Tätigkeiten wir generell als Projektmanagement verstanden. In der Praxis wird ein Projekt häufig in Teilprojekte gelgliedert, die jeweils auf eine bestimmte (Teil-) Zielerfüllung gerichtet sind und mit dem Hauptziel in Verbindung stehen. Dem Projektmanagement obliegt hierbei die Aufgabe die (Teil-) Projekte zu planen und zu steuern, sowie einer Überwachung im Hinblick auf zeitlich-, und ressourcenorientierte Parameter (z.B. die Mitglieder eines Teams).

Beim Projektmanagement entstehen vielfältige, detaillierte Informationen, die durch entsprechende Methoden gebündelt, kategorisiert und dem Management zur Bewertung präsentiert werden müssen, um Aussagen ableiten und Entscheidungen im Hinblick auf die Zielerreichung treffen zu können. Ohne eine entsprechende Technologie dauert es ab einem gewissen Komplexitätsgrad nicht lange, bis Projektmanagement mit händischer Arbeit nicht mehr zu bewältigen ist. Die Mitarbeiter eines Teams verlieren ohne eine geeignete Technologie zu viel Zeit für das Projektmanagement, und die Zielerreichung ist gefährdet.

Was ist ein Projektmanagement-System:

Eine Projektmanagement- Software ist ein Software, welche den Projektmitarbeitern eines Teams eine EDV-gestützte Hilfe bei ihren Projekttätigkeiten bietet. In der Projektmanagement-Software werden sämtliche Daten und Informationen des Projektes inklusive der vorhandenen Teilprojekte sowie aller Transaktionen und der damit verbundenen Kommunikation erfasst und in Datenbanken gespeichert. Dies können beispielhaft folgende Informationen sein:

- Dokumentation eines Meetings zur Zieldefinition einzelner Teilprojekte
- Budgetverwaltung je (Teil-)Projekt
- Terminplanung einzelner Projektschritte

Projektmanagement-Systeme basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße, dem jeweiligen Team und dem notwendigen Informationsgrad kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des Projektmanagement-System. Es kann daher vorkommen, dass die bisherigen Funktionen der Projektmanagement-Software mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

Vorteile und Nutzen eines Projektmanagement-Systems

Die Selbstorganisation eines Teams oder auch einer Einzelperson ist bei steigendem Informationsgrad eine enorme Herausforderung. Handgeschriebene Listen, hier und da einen Zettel oder ein Post-IT helfen nur bedingt weiter, und stellen sicherlich nicht die besten Methoden dar. Der Einsatz einer Projektmanagementsoftware kann bei bedachter Auswahl entscheidende Vorteile für die Projektarbeit bringen. Vor allem dann, wenn mehrere Personen an einem Projekt oder Teilprojekt beteiligt sind, und der Informationsfluss nochmals größer wird. Im Kern sollte jede Projektmanagementsoftware folgende Grundbereiche abdecken:

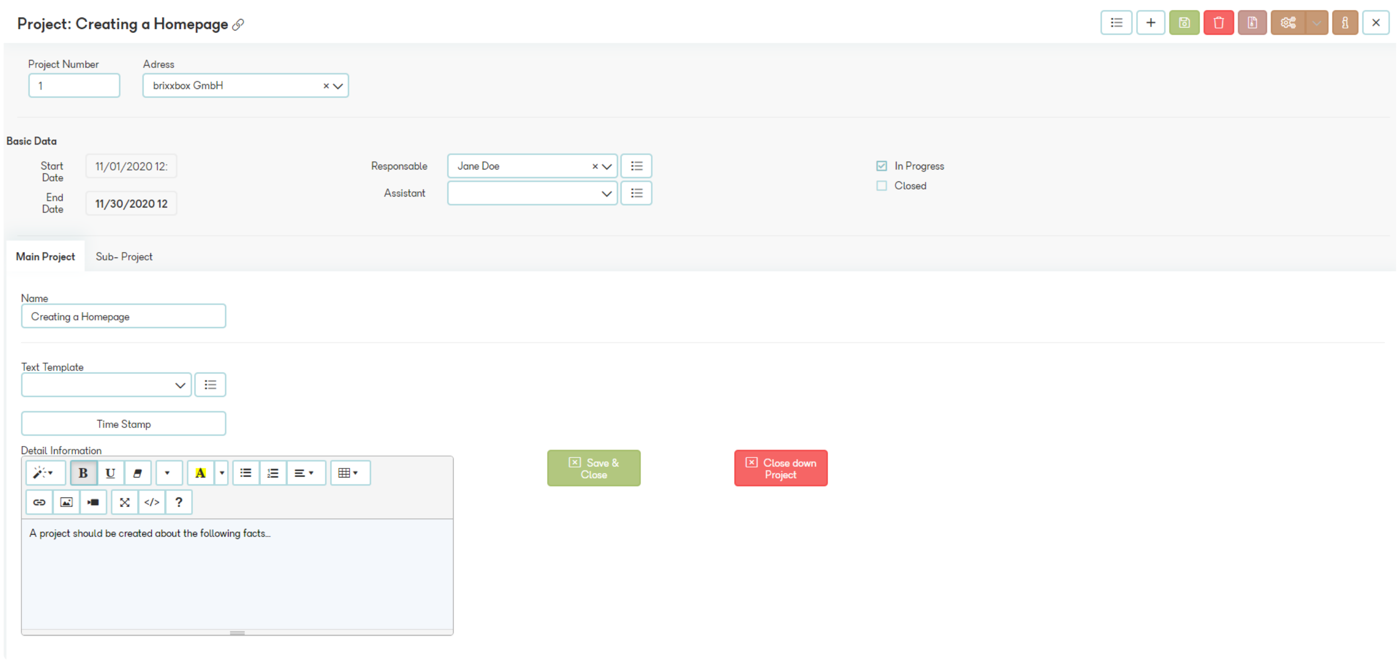
- Aufgabenverwaltung des Projektes
- Projektplanung einzelner Teilprojekte
- Projektsteuerung und Verwaltung der Ressourcen

Werden die zuvor genannten Bereiche durch die Software unterstützt, ist es jedem Projektteilnehmer und darüber hinaus dem Management möglich einen Überblick aller Ressourcen zu erhalten und die eigene Arbeit effizienter zu gestalten. Im Ergebnis liegen die Vorteile mit der richtigen Software auf der Hand. Es steigert die Möglichkeiten der Kontrolle, der Übersicht und der Transparenz aller Ressourcen. Eine Optimierung des Personaleinsatzes und eine höhere Kostenkontrolle sind in diesem Zusammenhang ebenfalls zu nennen.

brixxbox Low-Code-Plattform: Projektmanagement-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.



Project: Creating a Homepage

Project Number: 1
Adress: brixxbox GmbH

Basic Data
Start Date: 11/01/2020 12:
End Date: 11/30/2020 12
Responsable: Jane Doe
Assistant:
☑ In Progress
☐ Closed

Main Project | Sub- Project

Name: Creating a Homepage
Text Template:
Time Stamp
Detail Information: A project should be created about the following facts...

Save & Close
Close down Project

In der Praxis könnte diese Projektmanagement-Anwendung zum Beispiel von:

- Beratungsunternehmen
- Freelancern
- und Kleinunternehmen

genutzt werden, um Projekte anzulegen und zu verwalten. Neben der Stammdatenhaltung sind darüber hinaus erweiterte Funktionen enthalten, um eine mögliche Verwendung und Verknüpfung der Stammdaten zu visualisieren und einen realistischen Praxisbezug zur Unterstützung des Managements und anderer Abteilungen herzuleiten.

Auszug der Funktionen

- Unternehmensdaten: Bezeichnung und Anschrift der Unternehmen
- Team und Ansprechpartner: Kontaktdaten und Ansprechpartner der Unternehmen und Mitarbeiter
- Telefonbuch: auf Basis der Kontaktdaten
- Archivierung: Ablage und Digitalisierung von Dokumenten und Dateien von Kunden und Interessenten
- Vorlagen: Textvorlagen können erstellt werden zur vereinfachten Datenpflege
- Projekte: Anlage und Beschreibung von Projekten und zugehörigen Teilprojekten
- Phasen: Ausführliche Beschreibung und Dokumentation der Projekte und der jeweiligen Phasen
- Planung: Festlegung der Zuständigkeit innerhalb der Projekte und Teilprojekte
- Projekte überwachen
- Erstellen von projektrelevanten Erinnerungen und Zuweisung einzelner Mitarbeiter
- Zuordnung von Projekten zu Kunden, oder anderen Adressen
- Ressourcenverwaltung

Weitere Informationen finden Sie unter https://brixxbox.net

Ticketsystem

Contents

Was ist ein Ticketsystem

In der Praxis erreichen Unternehmen regelmäßig zahlreiche Anfragen. Kunden und Interessenten haben Fragen zu bestimmten Produkten und Diensteislungen, oder Klärungsbedarf bezogen auf die erworbenen Produkte. Ein Ticketsystem ist ein Softwaresystem, das zentralisiert alle Anfragen an ein Unternehmen verwaltet. Die Entgegennahme, die Koordination zur jeweils richtigen Abteilung oder Person und die Bearbeitung der Anfragen erfolgen zentral über das Ticketsystem und bündelt die Kanäle:

- Telefon
- E-Mail
- Facebook
- Instagram
- Chat und Messenger

in einem System. Die erhaltenen Anfragen können in der Folge geprüft und einer Person zur weiteren Bearbeitung zugewiesen werden bis zur Lösung (closed Ticket). Mit dem Ticketsystem soll sichergestellt werden, dass keine Nachricht verloren geht und jederzeit ein Gesamtüberblick über die zu bearbeitenden Vorgänge möglich ist.

Vorteile und Nutzen eines Ticketsystems:

Gute Ticketsysteme bieten vielfältige Möglichkeiten und unterstützen viele Prozesse im Unternehmen. Die in diesem Zusammenhang zu nennenden Hauptvorteile sind:

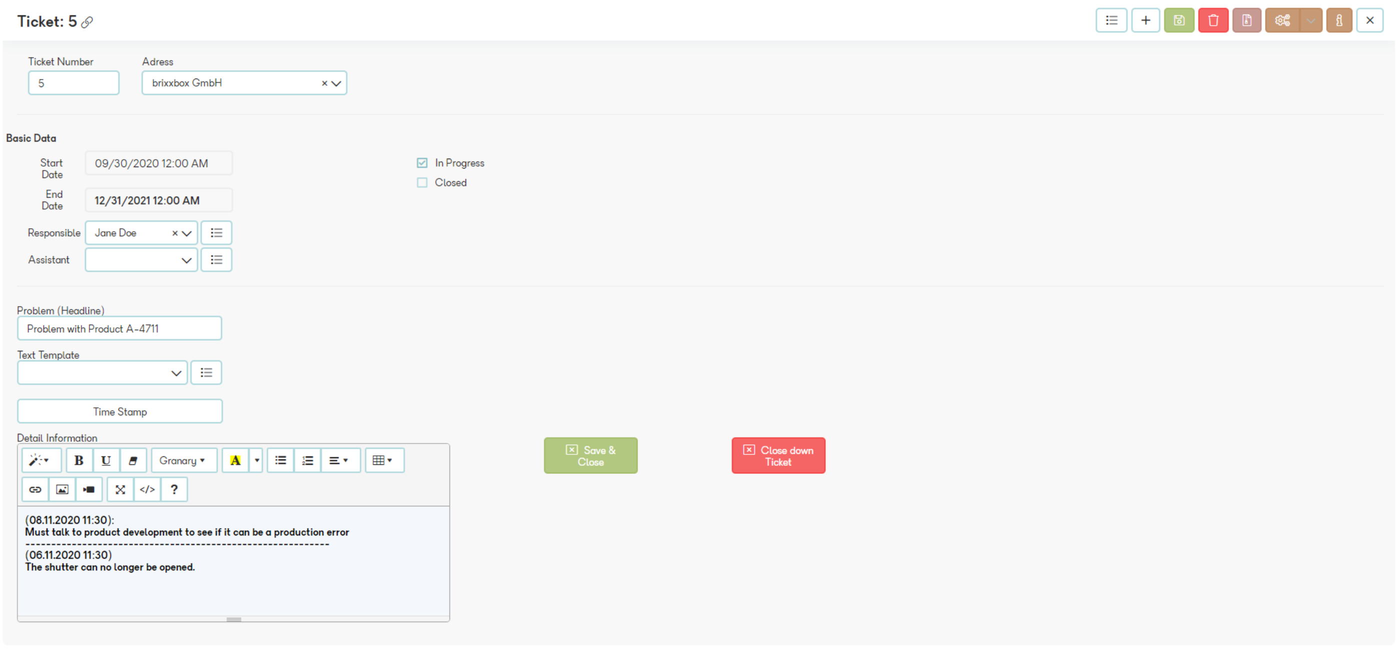
- Kundenanfragen straffen: Eingehende Tickets werden mit einem entsprechenden System aus mehreren Kommunikationskanälen an einem zentralen Ort gespeichert. Mitarbeiter können den Tickets Prioritäten zuweisen, sie überwachen und bearbeiten.
- Zugriff auf Kunden und Historie: Eine gute Beziehung zu Kunden und Partnern ist ein wichtiger Erfolgsfaktor für ein Unternehmen. Mit dem Ticketsystem erhalten Mitarbeiter Zugriff auch auf vergangene Kontakte mit den Kunden und Partnern, um stets alle Fakten im Blick zu haben
- Automatisierung von Aufgaben: Mit einem Ein Ticket-System können Routineaufgaben standardisiert werden, um Effizienz zu steigern und schnellere Antworten zu ermöglichen.
- Umfassende Auswertungsmöglichkeiten: Mit einem guten Ticketsystem können verschiedene Auswertungen schnell und einfach aufgerufen werden.

Ticketsysteme basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße, der Anzahl und dem Komplexitätsgrad der Anfragen kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des Ticketsystems. Es kann daher vorkommen, dass die bisherigen Funktionen des Ticketsystems mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

brixxbox Low-Code-Plattform: Ticketsystem-Template:

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.


Ticket: 5

Ticket Number: 5

Adress: brixxbox GmbH

Basic Data

Start Date: 09/30/2020 12:00 AM

End Date: 12/31/2021 12:00 AM

Responsible: Jane Doe

Assistant:

[✓] In Progress

[ ] Closed

Problem (Headline): Problem with Product A-4711

Text Template:

Time Stamp

Detail Information

(08.11.2020 11:30):
Must talk to product development to see if it can be a production error
----
(06.11.2020 11:30)
The shutter can no longer be opened.

In der Praxis könnte das Ticketsystem zum Beispiel von:

- Beratungsuntermehmen
- Freelancern
- und Kleinunternehmen

genutzt werden, um Anfragen aller Art zu verwalten und zu dokumentieren.

Auszug der Funktionen

- Firmenbezeichnung und Anschrift der Kunden
- Ansprechpartnerverwaltung mit den zugehörigen Kontaktdaten
- Integriertes Telefonbuch mit Suchfunktion
- Archivierung und Digitalisierung von Dokumenten und Dateien (DMS)
- Erfassung von Tickets
- Dokumentation von Kundenanfragen
- Nachverfolgung der Tickets durch eindeutige Nummerierung

Weitere Informationen finden Sie unter https://brixxbox.net

Digitale Adressverwaltung

Contents

Wer benötigt ein digitale Adressverwaltung:

Adressen, z.B. in Form von Kunden und Lieferanten, bilden für Unternehmen die Basis wirtschaftlichen Handelns. Eine sorgfältige und detaillierte Anlage und Verwaltung der jeweiligen Adressen und die damit verbundenen Ansprechpartner sind sehr wichtig. Gerade im Hinblick auf die Generierung und Bindung von Neukunden durch geeignete Akquisen fallen schnell viele Adressen an. Ohne eine unterstützende Technologie dauert es nicht lange, bis das Adressmanagement mit händischer Arbeit nicht mehr zu bewältigen ist.

Ein digitales Adressmanagement sollte mindestens folgende Punkte berücksichtigen:

- Speicherung aller adressrelevanter Informationen
- Speicherung der zugehörigen Ansprechpartner
- Speicherung der jeweiligen Kontaktinformationen

Mittlerweile zählt die Erfassung von zusätzlichen Adressinformationen, wie Interessen, Wünsche oder Vorlieben ebenso dazu. Im Idealfall wird im Laufe der Zeit ein 360- Grad Blick erreicht.

Vorteile und Nutzen einer digitalen Adressverwaltung

Alle Daten liegen an einem zentralen Ort. Dies reduziert den Wartungs- und Aktualisierungsaufwand. In der Praxis ist ein digitales Adressmanagement in den meisten Fällen mit anderen Systemen und Prozessen innerhalb einer Software aus dem Bereich "Warenwirtschaft oder ERP" verknüpft. Sie bilden so zusagen die Stammdaten für relevante Folgeprozesse wie:

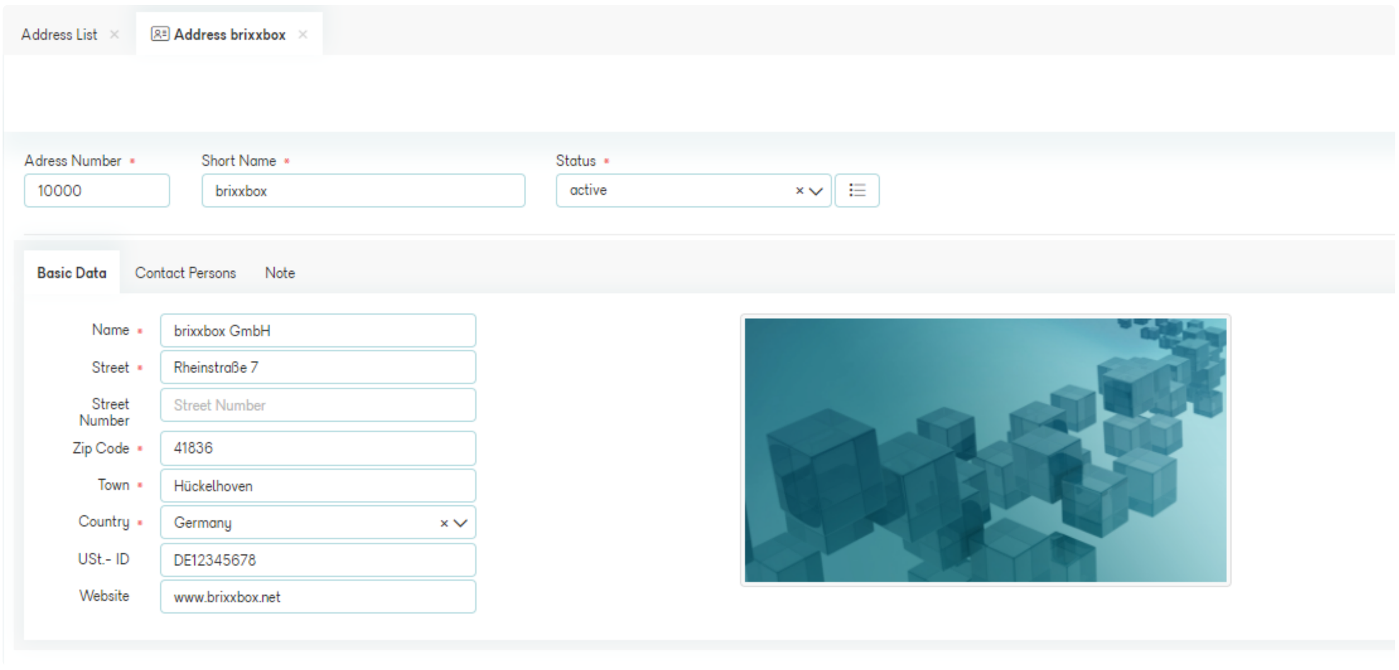
- Schreiben von Lieferscheinen und Rechnungen
- oder Verknüpfung zu CRM- relevanten Funktionen

Anwendungen zur digitalen Adressverwaltung in Verbindung mit relevanten Folgeprozessen basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße, der Ausrichtung und dem gewünschten Digitalisierungsgrad kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen der benötigten Lösung. Es kann daher vorkommen, dass die bisherigen Funktionen mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

brixxbox Low-Code-Plattform: digitale Adressverwaltung-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.



Adress Number *	10000
Short Name *	brixxbox
Status *	active
Name *	brixxbox GmbH
Street *	Rheinstraße 7
Street Number	Street Number
Zip Code *	41836
Town *	Hückelhoven
Country *	Germany
USt.- ID	DE12345678
Website	www.brixxbox.net

In der Praxis könnte die digitale Adressverwaltung zum Beispiel von:

- Beratungsuntermehmen
- Freelancern
- und Kleinunternehmen

genutzt werden, um nach der Firmengründung erste Kontakte und die jeweiligen Ansprechpartner zu verwalten.

Auszug der Funktionen

- Firmenbezeichnung und Anschrift der Kunden, Lieferanten, Interessenten und sonstigen Adressen
- Ansprechpartnerverwaltung mit den zugehörigen Kontaktdaten
- Integriertes Telefonbuch mit Suchfunktion
- Archivierung und Digitalisierung von Dokumenten und Dateien (DMS)
- Dokumentation von adressrelevanten Besonderheiten (z.B. bestimmte Lieferfristen, Annahmefristen, abweichende Bedingungen etc.)

Weitere Informationen finden Sie unter https://brixxbox.net

Fahrtenbuch

Contents

Wer benötigt ein digitales Fahrtenbuch:

Ein Fahrtenbuch hat grundsätzlich den Zweck Fahrstrecken mit einem Fahrzeug sowie den jeweiligen Reisegrund zu dokumentieren. Verwenden Selbstständige oder Angestellte ein auf das Unternehmen angemeldete Auto, geht das Finanzamt grundsätzlich davon aus, dass die zuvor genannten Personengruppen das Fahrzeug nicht nur für Fahrten des Unternehmens, sondern auch für private Angelegenheiten nutzen. Der durch das Auto entstehende geldwerte Vorteil muss in den meisten Fällen versteuert werden.

Dazu stehen zweierlei Optionen zur Verfügung: Entweder wird z.B. das Auto mit einer Pauschale gemäß der Ein-Prozent-Regelung veranschlagt – oder es wird die Häufigkeit und Dauer aller dienstlichen und privaten Fahrten mit dem Auto in einem ausführlichen Fahrtenbuch dokumentiert. Die letztere Methode ist zwar mit einigem Aufwand verbunden, kann jedoch attraktive Steuervergünstigungen bescheren. Dafür müssen jedoch die Anforderungen und die deklarierten Angaben der Finanzbehörde eingehalten halten:

- Für jedes Fahrzeug muss ein eigenes Fahrtenbuch geführt werden
- Angaben zum amtlichen KfZ-Kennzeichen des betreffenden Fahrzeugs
- Angabe des jeweiligen Fahrers
- Angaben zum genauen Kilometerstandes zu Beginn der Fahrt und zum genauen Kilometerstand zum Ende der Fahrt, sowie der gefahrenen Kilometer
- Dokumentation des Zeitraums der Fahrt

Bei der manuellen Erfassung und Angabe der einzelnen Fahrten entsteht schnell ein großer Aufwand. Ohne eine unterstützende Technologie dauert es nicht lange, bis die Verwaltung mit händischer Arbeit nicht effizient ist.

Vorteile und Nutzen eines digitalen Fahrtenbuches

Die Nutzung eines elektronischen Fahrtenbuches bietet viele Vorteile und führt häufig zu geringeren Kosten, gerade bei vielen Fahrten für das Unternehmen und wenig privaten Fahrten. Generell lassen sich folgende Vorteile nennen:

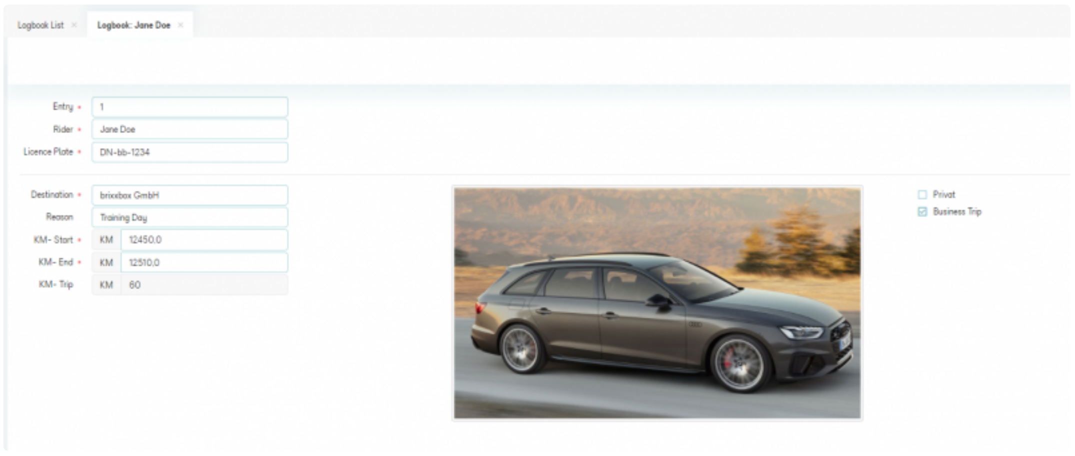
- deutliche Zeitersparnis,
- Klare Übersicht und kaum Fehler
- Sicherheit gegenüber dem Finanzamt,
- In der Regel geringere Kosten und Steuervorteile gegenüber der Ein-Prozent-Regelung,
- elektronische Auswertung der Fahrdaten.

Anwendungen zur Verwaltung digitaler Fahrtenbücher basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße, der Anzahl der Firmenfahrzeuge und dem Grad der Internationalisierung kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des digitalen Fahrtenbuches. Es kann daher vorkommen, dass die bisherigen Funktionen mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

brixxbox Low-Code-Plattform: digitales Fahrtenbuch-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.



Entry *	1
Rider *	Jane Doe
Licence Plate *	DN-bb-1234
Destination *	brixxbox GmbH
Reason	Training Day
KM- Start *	KM 12450.0
KM- End *	KM 12510.0
KM- Trip	KM 60

☐ Privat
☑ Business Trip

In der Praxis könnte das digitale Fahrtenbuch zum Beispiel von:

- Freelancern

genutzt werden, um das vorhandene Firmenfahrzeug digital zu verwalten, die jeweiligen Fahrten mit allen Angaben zu dokumentieren und bei Bedarf auszuwerten.

Auszug der Funktionen

- Dokumentation des Fahrers, des amtlichen Kennzeichens
- Erfassung der Kilometer beim Start und bei Beendigung der Fahrt sowie Ermittlung der gefahrenen Kilometer
- Unterscheidung zwischen geschäftlichen und privaten Fahrten im Hinblick auf die Versteuerung
- Nutzung als Handy App möglich
- Nutzung als App für Tablets möglich

Weitere Informationen finden Sie unter https://brixxbox.net

Auftragsverwaltung

Contents

Wer benötigt ein digitale Auftragsverwaltung:

Ein Geschäftsmodell wurde entwickelt und der Markt sondiert. Im nächsten Schritt wurden die Produkte und Dienstleistungen deklariert und erste Adressen für Akquisen besorgt. Im Idealfall werden in der Folge die ersten Kunden sowie die ersten Kundenaufträge generiert. Die Geschäftsprozesse nehmen zu. Ohne eine unterstützende Technologie dauert es nicht lange, bis das Arbeitsaufkommen mit händischer Arbeit nicht mehr zu bewältigen ist.

Eine Software zur digitalen Auftragsverwaltung ist eine sinnvolle Lösung, sofern Sie zumindest folgende Punkte berücksichtigt:

- Speicherung aller adressrelevanter Informationen inklusive Ansprechpartner und Kontaktinformationen in der Software
- Speicherung der vorhandenen Artikel und Dienstleistungen in der Software
- Anlage der Kundenaufträge und Statusverwaltung in der Software

Mittlerweile zählt die Erfassung von zusätzlichen Informationen, wie besondere Lieferorte, bestimme Anlieferzeiten oder andere Vorlieben ebenso dazu.

Vorteile und Nutzen einer digitalen Auftragsverwaltung

Bei einem solchen System liegen alle Daten an einem zentralen Ort. Adressen, Artikel und sonstige Stammdaten können in der Software effektiv verwaltet und bei Bedarf aktualisiert werden. In der Praxis ist eine digitale Auftragsverwaltung in den meisten Fällen mit anderen Systemen und Prozessen innerhalb einer Software aus dem Bereich "Warenwirtschaft oder ERP" verknüpft. Dazu zählen z.B.:

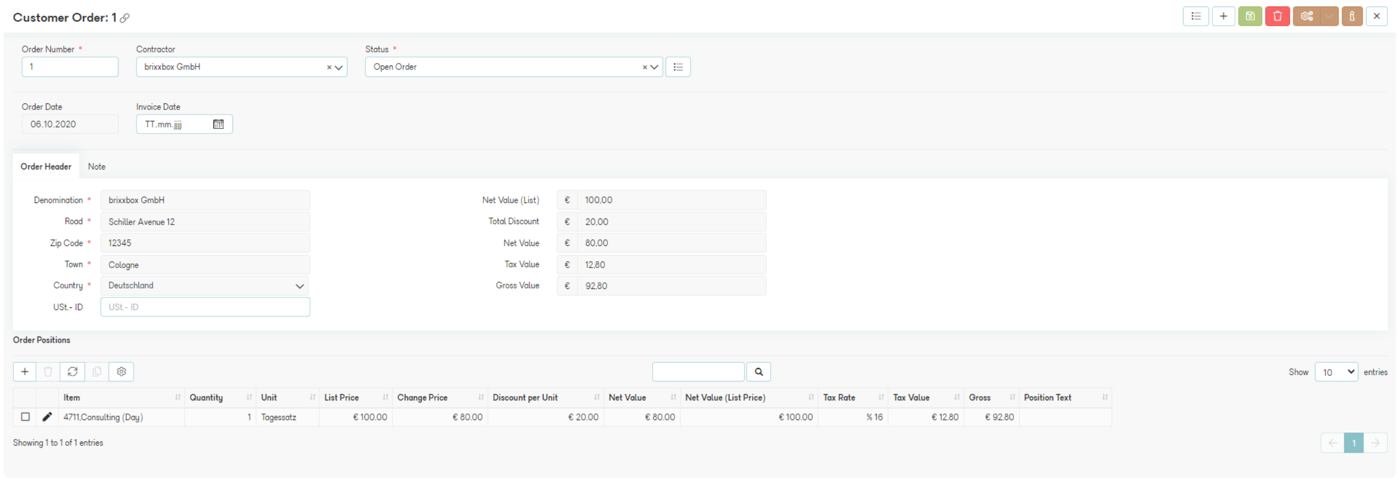
- Abrechnungsprogrammen
- CRM-Systemen, um die Kundenwünsche direkt zu berücksichtigen
- Archivsystem, um Dokumente digital zu archivieren

Anwendungen zur digitalen Auftragsverwaltung in Verbindung mit relevanten Folgeprozessen basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße, der Ausrichtung und dem gewünschten Digitalisierungsgrad kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen der benötigten Lösung. Es kann daher vorkommen, dass die bisherigen Funktionen mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

brixxbox Low-Code-Plattform: digitale Auftragsverwaltung-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.


Customer Order: 1

Order Number *	Contractor	Status *
1	brixxbox GmbH	Open Order
Order Date	Invoice Date	
06.10.2020	TT.mm.jjjj	

Order Header | Note

Denomination *	brixxbox GmbH	Net Value (List)	€ 100,00
Road *	Schiller Avenue 12	Total Discount	€ 20,00
Zip Code *	12345	Net Value	€ 80,00
Town *	Cologne	Tax Value	€ 12,80
Country *	Deutschland	Gross Value	€ 92,80
USt.- ID	USt.- ID		

Order Positions

Show 10 entries

Item	Quantity	Unit	List Price	Change Price	Discount per Unit	Net Value	Net Value (List Price)	Tax Rate	Tax Value	Gross	Position Text
4711.Consulting (Day)	1	Tagessatz	€ 100.00	€ 80.00	€ 20.00	€ 80.00	€ 100.00	% 16	€ 12.80	€ 92.80	

Showing 1 to 1 of 1 entries

In der Praxis könnte die digitale Auftragsverwaltung zum Beispiel von:

- Beratungsuntermehmen
- Freelancern
- und Kleinunternehmen

genutzt werden, um nach der Firmengründung erste Kontakte mit den jeweiligen Ansprechpartnern zu verwalten sowie erste Geschäftsaktivitäten (z.B. Kundenaufträge) überwachen.

Auszug der Funktionen

- Firmenbezeichnung und Anschrift der Kunden, Lieferanten, Interessenten und sonstigen Adressen
- Ansprechpartnerverwaltung mit den zugehörigen Kontaktdaten
- Archivierung und Digitalisierung von Dokumenten und Dateien (DMS)
- Dokumentation von adressrelevanten Besonderheiten (z.B. bestimmte Lieferfristen, Annahmefristen, abweichende Bedingungen etc.)
- Artikelverwaltung
- Preisverwaltung
- Erfassung und Steuerung von Rabatten je Auftrag
- Auftrag eingeben und dokumentieren
- Status des Auftrags verwalten

Weitere Informationen finden Sie unter https://brixxbox.net

Basis CRM

Contents

Was ist CRM

Das Customer Relationship Management/ Kundenbeziehungsmanagement (kurz CRM) ist für Unternehmen und deren Mitarbeiter eine Strategie zur systematischen Gestaltung von Beziehungen und Interaktionen mit Kontaktpersonen bestehender und potenzieller Kunden. Diese sind durch die Umsatzgenerierung maßgeblich am langfristigen Erfolg des Unternehmens beteiligt. Wenn Kunden Ihr Kaufverhalten anpassen und auf Produkte anderer Unternehmen ausweichen, kann dies für das eigene Unternehmen ernste Folgen haben. Daher investieren Unternehmen viel Zeit und Aufwand in die CRM-Tätigkeiten zur Analyse der Kundenwünsche. Zielgruppen, Märkte und die jeweiligen Bedürfnisse werden im Rahmen des CRM analysiert und mit dem eigenen Portfolio abgestimmt. Werden Abweichungen festgestellt, obliegt den Mitarbeitern des Vertriebs und des Marketings in Abstimmung mit dem Management die Aufgabe das Kundenverhältnis im Rahmen des CRM zu Optimieren bzw. die konkreten CRM-Strategien anzupassen, um die Kunden wieder besser zu erreichen. Auch andere Abteilungen tragen zur Kundenzufriedenheit bei und liefern wertvolle Informationen für das CRM. Hierzu zählt zum Beispiel der gesamte Bereich Service. Im Rahmen des Service werden Kundenanfragen und Unstimmigkeiten erfasst und bearbeitet. Auch dies ist Teil des CRM.

Beim CRM entstehen vielfältige, detaillierte Informationen, die gebündelt, kategorisiert und bewertet werden müssen, um Aussagen ableiten zu können. Ohne eine entsprechende Technologie dauert es bei steigenden Kunden-, und Interessentenzahlen und den jeweiligen Kontakten nicht lange, bis CRM mit händischer Arbeit nicht mehr zu bewältigen ist. Der zielgerichtete Kontakt zum Kunden wird erschwert. Die Mitarbeiter verlieren ohne eine geeignete Technologie zu viel Zeit für das CRM, wenn die CRM-Strategie auch bei steigenden Kunden und Interessenten mit gleicher Qualität und Intensität verfolgt werden soll.

Was ist ein CRM-System

Ein CRM-System ist eine Software mit Funktionen zur Verwaltung und Automatisierung aller Informationen, die bei der Kunden-, bzw. Interessentenansprache und deren Bindung Monat für Monat anfallen. In der CRM-Software werden sämtliche Daten von Kunden und Interessenten inklusive aller Transaktionen und der damit verbundenen Kommunikation erfasst und in Datenbanken gespeichert. Dies können beispielhaft folgende Informationen sein:

- Dokumentation eines Interessententelefonates zur späteren Bewertung
- Ablage eines Kundenangebotes zur späteren Zuordnung
- Terminerinnerung für eine Angebotsbesprechung bei einem Interessenten vor Ort
- Geburtstagsverwaltung der A- und B-Kunden zur Aufmerksamkeitssteigerung

CRM-Systeme basieren überwiegend auf Standardsoftware-Produkten. Abhängig von der Unternehmensgröße und dem notwendigen Informationsgrad kommen verschiedene Lösungen mit unterschiedlichem Funktionsumfang in Frage. Wächst ein Unternehmen mit der Zeit, wachsen auch die Anforderungen und somit die notwendigen Funktionen des CRM-System. Es kann daher vorkommen, dass die bisherigen Funktionen der CRM-Software mit der Zeit nicht mehr ausreichen und ein Wechsel erwogen werden muss, um den steigenden Informationsgrad für das Unternehmen zu berücksichtigen.

Vorteile und Nutzen eines CRM-Systems

Die Vorteile und der Nutzen einer CRM-Software für Unternehmen liegen klar auf der Hand. In der Praxis muss eine Vielzahl von Kontakten und deren Beziehung zum eigenen Unternehmen in einem System verwaltet werden. Gut durchdachte CRM-Systeme schaffen mehr Transparenz. Sie können überall dort eingesetzt werden, wo direkter oder indirekter Kunden-, bzw. Interessentenkontakt erforderlich ist. Mit der richtigen CRM-Software kann ebenfalls der gesamte Außendienst unterstützt werden. Dies gilt sowohl für:

- den vertrieblichen Außendienst mit der Zielsetzung "Umsatzsteigerung",
- als auch für den serviceorientierten Außendienst mit der Zielsetzung "Service und Wartung".

Alle Aktivitäten vor Ort können direkt in die CRM-Software erfasst werden. Darüber hinaus kann ein gutes CRM-System auch eine Hilfestellung für bestimmte Tätigkeiten sein, in dem Informationen an der richtigen Stelle in der richtigen Form aufbereitet zur Verfügung stehen. Dies spart nicht nur Zeit, sondern liefert regelmäßig auch bessere Resultate in allen Bereichen.

Zusammengefasst trägt ein CRM-System dazu bei den Vertrieb, das Marketing, den Service und nicht zuletzt das Management zu unterstützen. Mit Hilfe eines CRM-Systems können Tätigkeiten optimiert und automatisiert (z.B. monatliche Reports) werden. Darüber hinaus liefert ein gutes CRM vielfältige Entscheidungsgrundlagen.

brixxbox Low-Code-Plattform: Basis CRM-Template

Mit der Cloud-Software "brixxbox" können Unternehmen Prozesse einfach und schnell digitalisieren. Durch das zugrunde liegende Baukastenprinzip können notwendige Anpassungen und Erweiterungen auf Wunsch des Managements oder anderer Mitarbeiter jederzeit in der Software vorgenommen werden, um zusätzliche Daten im System zu verarbeiten. Anpassungen im System beeinträchtigen in keiner Weise den Produktivbetrieb der Software im jeweiligen Unternehmen.

Die zur Verfügung gestellten Templates dienen als Konfigurationsbeispiel. Es sollen verschiedene Ansätze illustriert werden. Die Templates können nach dem Import beliebig ergänzt werden, oder als Vorlage für eigene Applikationen oder ganze Systeme verwendet werden.

In der Praxis könnte diese CRM-Anwendung zum Beispiel von:

- Freelancern
- und Kleinunternehmen

genutzt werden, um Kunden und Interessenten anzulegen und deren Kontakte und CRM-relevante Daten zu verwalten. Neben der Stammdatenhaltung sind darüber hinaus erweiterte Funktionen enthalten, um eine mögliche Verwendung und Verknüpfung der Stammdaten zu visualisieren und einen realistischen Praxisbezug zur Unterstützung des Managements, des Vertriebs und des Marketings herzuleiten.

Auszug der Funktionen

- Unternehmensdaten: Bezeichnung und Anschrift der Unternehmen
- Kontaktdaten: Kontaktdaten und Ansprechpartner der Unternehmen und Mitarbeiter
- Telefonbuch: auf Basis der Kontaktdaten
- Archivierung: Ablage und Digitalisierung von Dokumenten und Dateien von Kunden und Interessenten
- CRM Daten: Terminplanung (z.B. pro Monat, Woche), Terminüberwachung und Termindokumentation sowie Zuordnung relevanter Kontakte
- CRM Workflows: Erstellung von Erinnerungen und Wiedervorlagen (z.B. in zwei Monaten) für bestimme Kontakte
- Vorlagen: Textvorlagen können erstellt werden
- Mobil: Zugriff auf alle Daten überall und mit jedem Endgerät möglich (Cloud)

Weitere Informationen finden Sie unter https://brixxbox.net

Apps

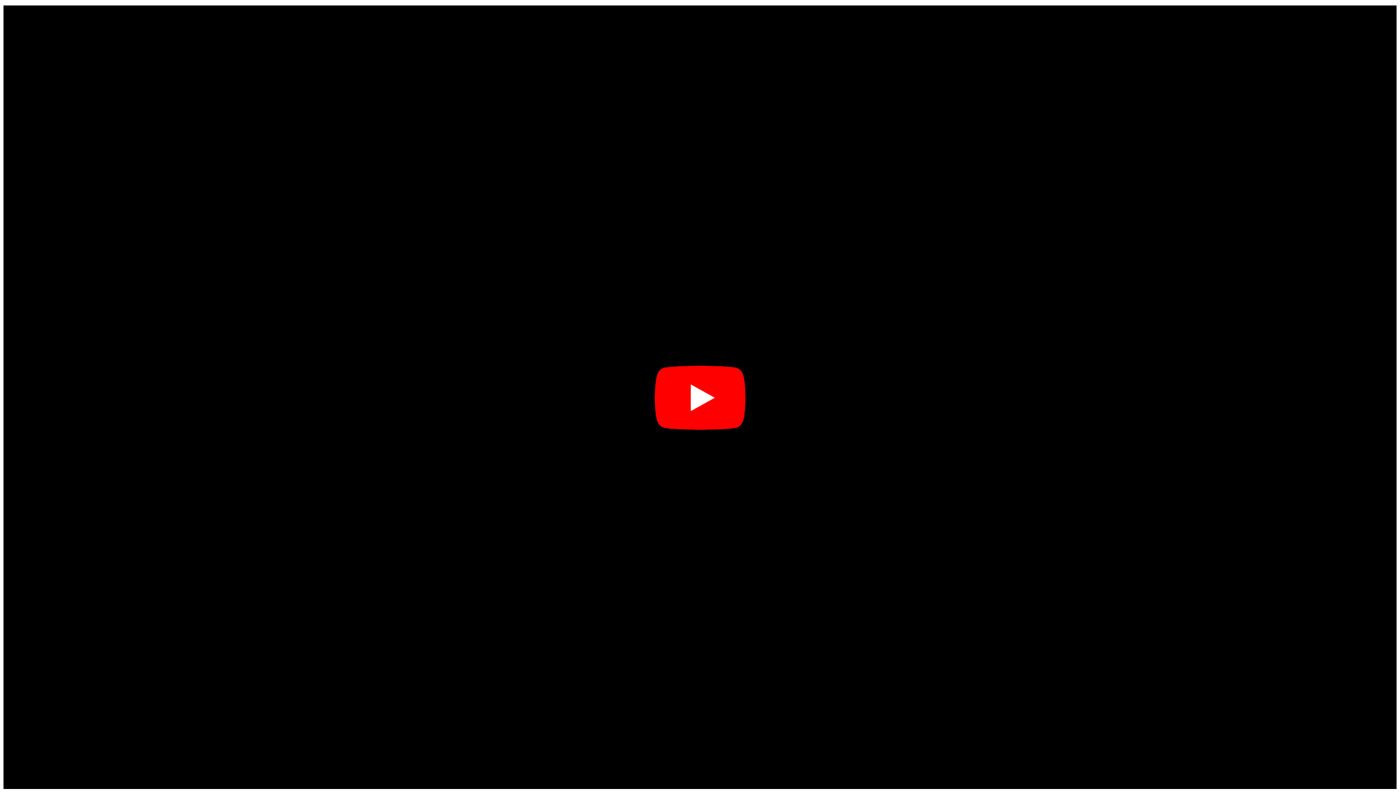
ApiKeys

This app lets you create or delete api access keys.

Installation of cloud Connector for Printing

> This is not the cloudGateway!

See how to use the Cloud Connector for printing.



Upcoming Features

- Run As a Service
- See Connector Online Status in Api Keys Panel
- Edit Config in Api Keys Panel

Docker Installation and Configuration

Beside the Print Gateway (as ssen in the video), there is a new docker gateway. Use the Docker command column to pull and start a Docker container in your local network.
Use the settings in the ApiKey list to configure the service endpoints for each docker installation like in the sample below.

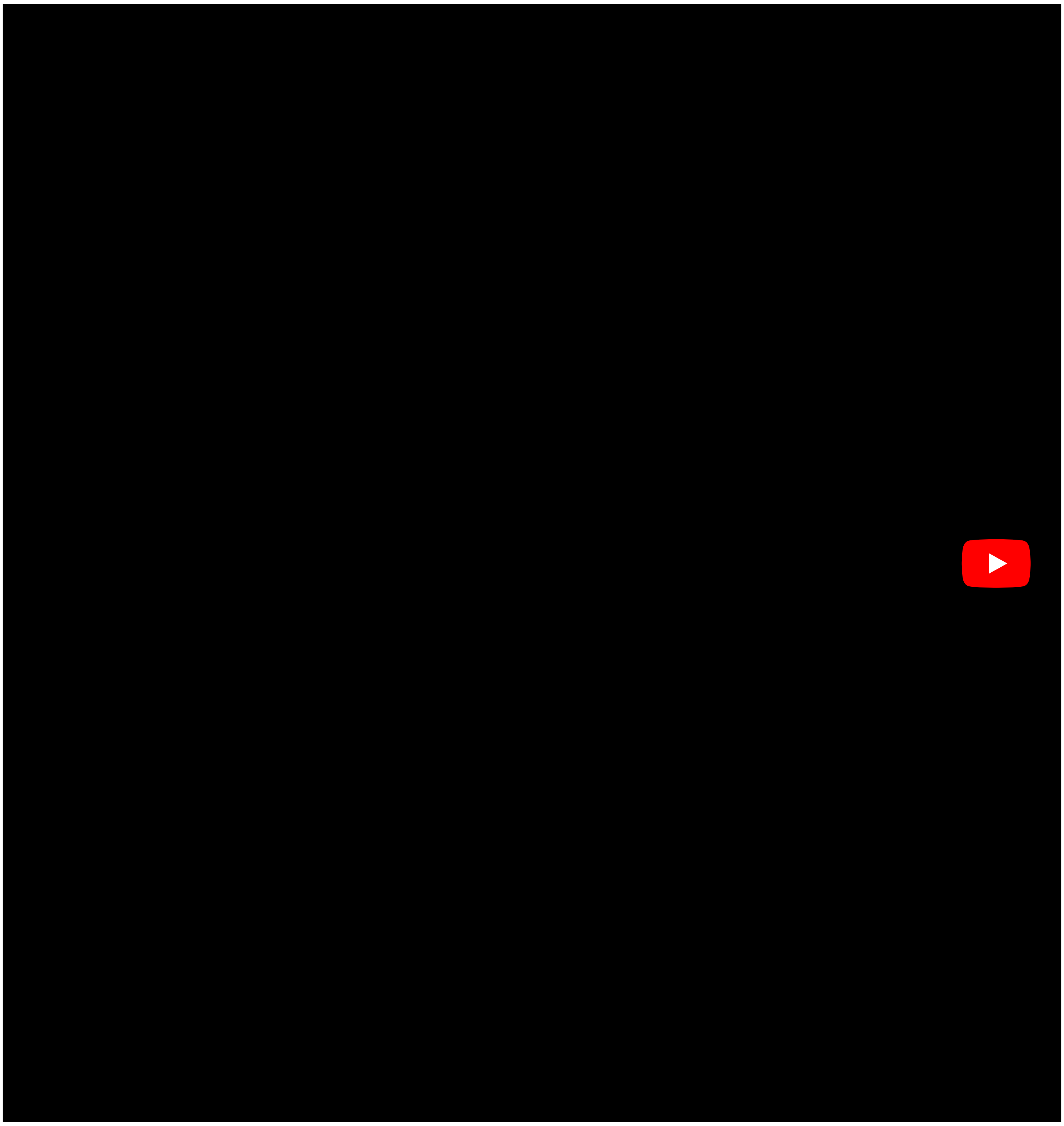
```
{
  "Connections": [
    {
      "Endpoint": "LocalFirebird",
      "Plugin": "Firebird",
      "ConnectionString": "Server=192.168.178.39;User=SYSDBA;Password=abcde"
    },
        {
      "Endpoint": "LocalSql",
      "Plugin": "MsSql",
      "ConnectionString": "Server=acme.com"
    }

  ]
}
```

CloudGateway

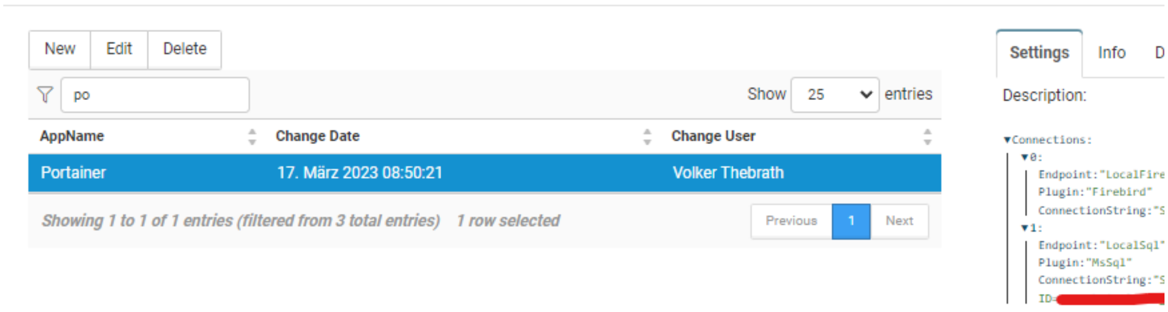
For Printing, use ApiKeys CloudeConnector

The CloudGateway is similar to the CloudConnector but the CloudGateway is used for retrieving and sending data instead of printing.

To request Data from the Gateway use the cloudQuery api function.



Example Configuration



po

Show 25 entries

AppName	Change Date	Change User
Portainer	17. März 2023 08:50:21	Volker Thebrath

Showing 1 to 1 of 1 entries (filtered from 3 total entries) *1 row selected*

Previous 1 Next

Description:

```
▼Connections:
  ▼0:
    Endpoint:"LocalFire
    Plugin:"Firebird"
    ConnectionString:"S
  ▼1:
    Endpoint:"LocalSql"
    Plugin:"MsSql"
    ConnectionString:"S
    ID=[redacted]
```

```
{
  "Connections": [
    {
      "Endpoint": "LocalFirebird", //Your name choice
      "Plugin": "Firebird",
      "ConnectionString": "Server=192.168.178.39;User=SYSDBA;Password=xxxxxxxxxxxxxx;Database=C:\\Users\\volker\\Documents\\VOLKE
    },
        {
      "Endpoint": "LocalSql",
      "Plugin": "MsSql",
      "ConnectionString": "Server=tcp:192.168.178.39,1433;Initial Catalog=MyDb;Persist Security Info=False;User ID=myUser;Passwor
    }
  ]
}
```

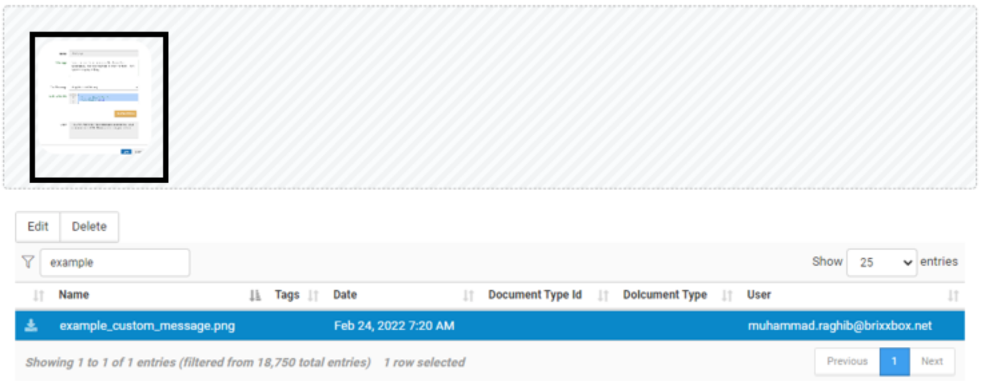
Attachments

Attachments

In this app Brixxbox allow users to add any number of different file types to Brixxbox workspace from outside. It also allows all the documents generated within the workspace to be stored in this app. These documents can be used to facilitate the proper functioning of the user systems like generating reports for system user for example: report on total number of order placed and also for end user for example: it can be an invoice gernerated for a customer.All of these documents can be stored under this app. User must assign document type to each document.

Tutorial

We want to add an example image from our pc to brixxbox. Brixxbox allow users two ways to add a file: via drag and drop files from pc to attachments page, and via opening a file explorer then selecting a specific file. Lets now add our image.



Name	Tags	Date	Document Type Id	Dolcument Type	User
example_custom_message.png		Feb 24, 2022 7:20 AM			muhammad.raghib@brixxbox.net

Showing 1 to 1 of 1 entries (filtered from 18,750 total entries)   1 row selected

As we can see from above snapshot that added image is visible in the drag and drop area(highlighted in black). Also if you search in the list of attachments, you can find the same image. The interesting fact to consider here is that when we upload the file only file name, upload time, and user is being set. The properties document type and id are left as it is. It gives user an option to select the document type by himself. Lets do this by selecting the file and then clicking edit button. An edit panel will open, now select the "bild" type as we have selected file of type image and click on update. In this easy tutorial we learned how to add a file in Brixxbox and assign a document type to it.

AuditHistory

Record History

The Record or audit) history allows you to monitor all the changes to records and restore previous states for single records.

Demo

DevelopmentChangeHistory

This App will provide an overview for the changes done in the current workspace. The default view will list the changes chronologically and show the different types of changes.

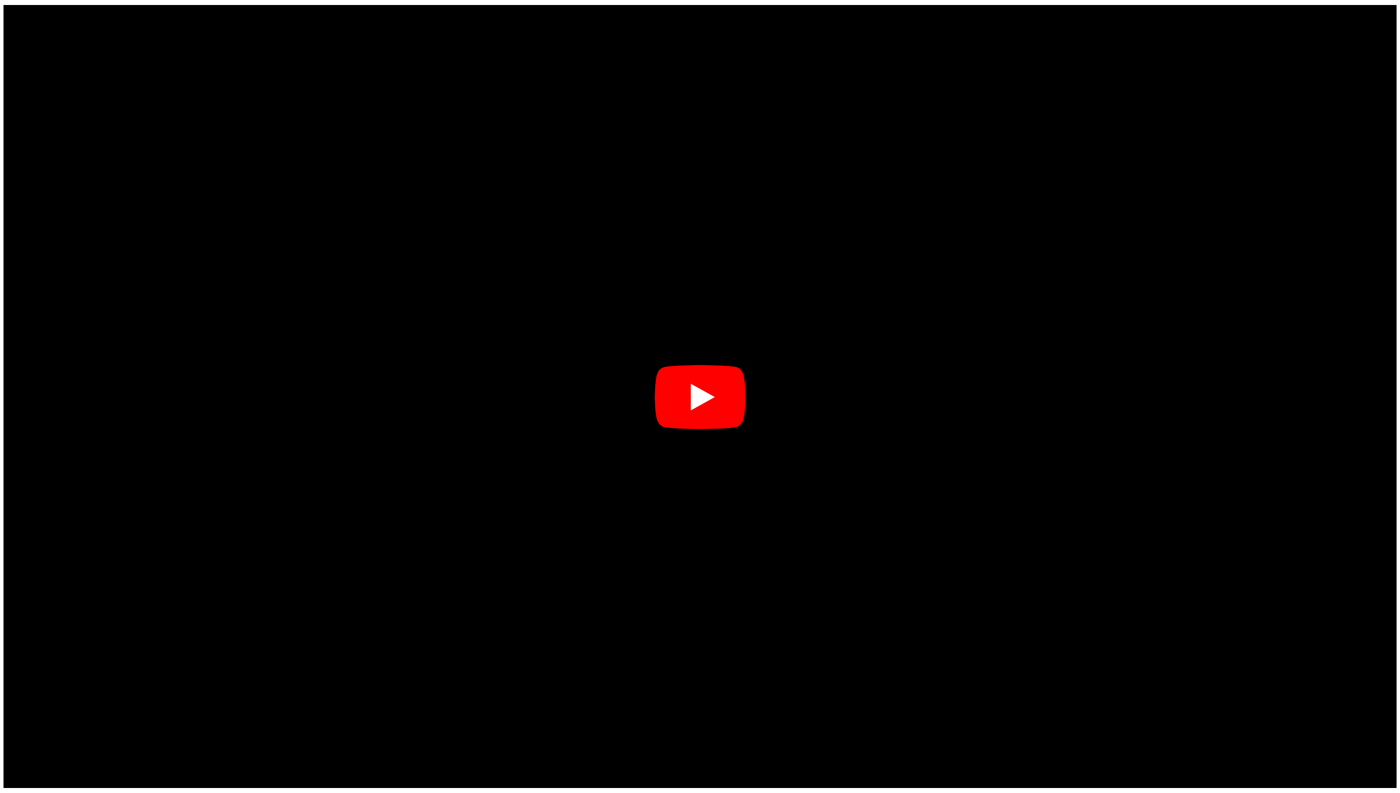
Provided the necessary access rights a selection of other workspaces is shown. Marked lines within the change history can be compared against the same items in other workspaces. If the comparison differs, a view can be triggered to show the differences in detail.

DocumentTypes

Document Types

Here in the demom below we provide with all the necessary information for adding, editing, and using different document types inside our workspace.

Demo



Jobs

In this app, Brixxbox allow users to add automated jobs to their workspace. If a user want to perform a task at regular intervals then user can add its task in a job schedular provided by Brixxbox.

Job

Job provides functionality to automate the tasks provided by user after specific time interval and on regular basis. User can create one job for each task. Brixxbox will provide book keeping of those jobs, for example logging of errors etc. A job contains following properties.

Job Properties

Name

Each job should have a meaningful name. It will help user to identify the purpose of the job. For example, status update job etc.

Cron

It is one of the most important properties. It is used to schedule jobs, it specifies that the job will run on a given scheduled time. Its value is divided in to five parts(* * * * *). The first part represent minutes. The allowed values ranges from(0-59), The second one represent hours and its value ranges from(0-23). Third part is day of a month, its value ranges from (1-31) . The fourth part is month itself, its value ranges from(1-12). The last part represents day of the week(0-6). 0 is for sunday, 1 is for monday and so on. In order to learn more about CRON click here.

Type

App job allows user tasks to be of three types: server side function click here, sql statement click here, and sql stored procedure click here.

Script Select

It is the customized code for above job types. It can be stored in the Brixxbox and will be utlized here.

User id

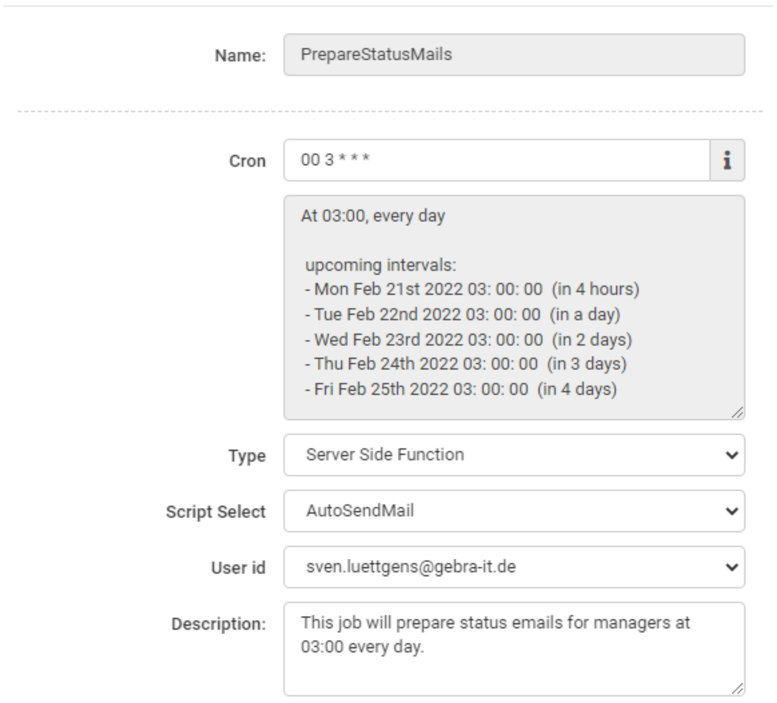
Each job should have a user id attached to it. This creates ownership of the app job.

Description

Each job should have have a meaningful description. It is optional but it will help all the workspace users to know exactly what a job is doing.

Example

Lets now look at one example job. Suppose we want to prepare status emails for managers. We want to generate these emails at 03:00 every day. For this purpose we need to set the CRON value to (00 3 * * *). It means that on 00 minute of 03:00 hour, on every day of each month and on each week day execute this job. We also want to make this server side function. In next step, we will choose custom code script, as we have already written this script, we can select it from drop down list. Now we need to assign a user id to this job. In the end, we need to give description for our newly created job. In the below snapshot, we can see all the options filled. Click save button to save this job.



Name:	PrepareStatusMails
Cron	00 3 * * *
	At 03:00, every day upcoming intervals: - Mon Feb 21st 2022 03: 00: 00 (in 4 hours) - Tue Feb 22nd 2022 03: 00: 00 (in a day) - Wed Feb 23rd 2022 03: 00: 00 (in 2 days) - Thu Feb 24th 2022 03: 00: 00 (in 3 days) - Fri Feb 25th 2022 03: 00: 00 (in 4 days)
Type	Server Side Function
Script Select	AutoSendMail
User id	sven.luettgens@gebra-it.de
Description:	This job will prepare status emails for managers at 03:00 every day.

Cron Definitions

Cron Expression

Basic structure

A cron expression consists of five individual parts (That is the basic version. Some cron expressions support additional values).

Part	Allowed Values	
Minutes	0-59	
Hours	0-23	
Day of month	1-31	
Month	1-12	
Day of week	0-6 (0 = Sun, 1 = Mon, ...)	

In addition to the above values the following characters can be used.

Character	Meaning	
`*`	**Any value** e.g. * as the value for `Hour` will select all the possible values 0,1,2,..,23 "every hour".	
`,`	**value list separator** e.g. 0,30 as the value for `Minute` will select 0 and 30 "every half hour".	
`-`	**Range of values** e.g. 1-5 as the value for `Day of week` will select Monday thru Friday	
`/`	**step values** e.g. */3 as the value for `Month` will select Jan, Apr, Jul, Oct. Writing "1,4,7,10" would yield the same result.	

examples

- `15 1 * * *` every day at 1:15 am
- `0 8-17 1-5 * *` Monday thru Friday, once every hour between 8am and 5:59pm
- `0 5 1 */3 *` at 5am on the first day of every third month of the year (Jan, Apr, Jul, Oct)
- `0 0 1 1 *` start of the year
- `*/5 3-14 * * 6` every five minutes between 3am and 2:59pm only on a Saturday

Cron expressions are widely used. To get more detailed information those links might be helpful

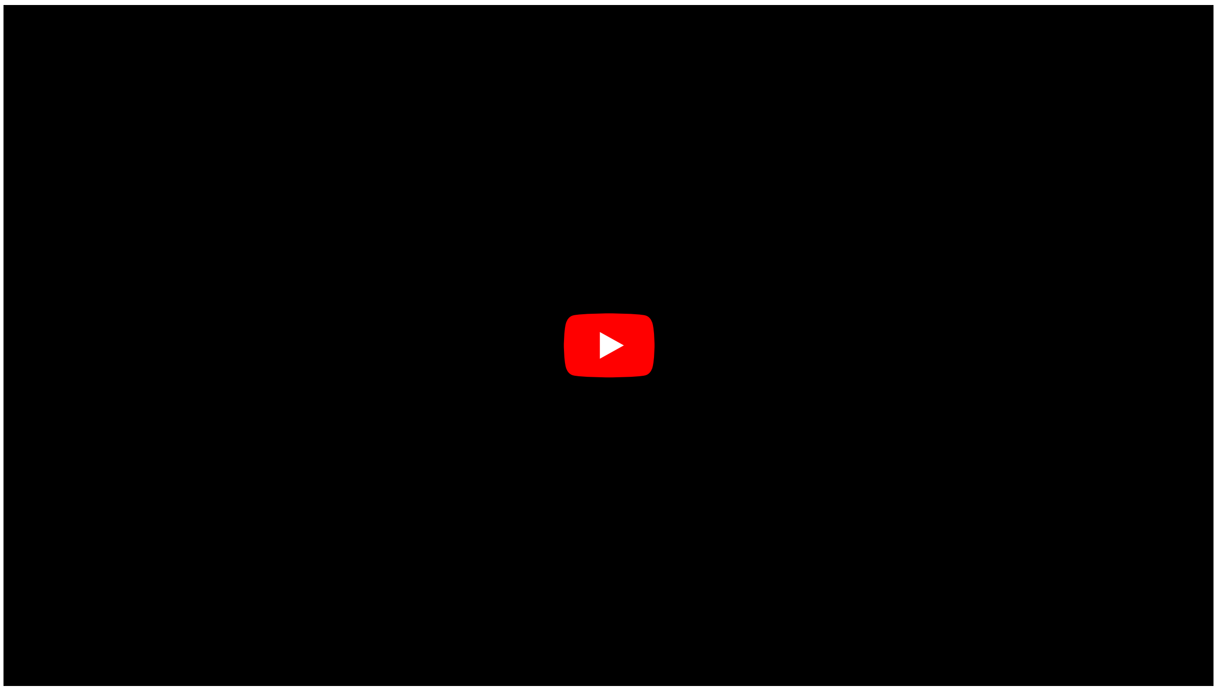
- https://en.wikipedia.org/wiki/Cron
- https://crontab.guru

Menu Editor

Menu Editor

The brixxbox menu for a workspace can be configured here. The menu is a tree like structure with folders and app endpoints in it.

Demo



How to edit the menu

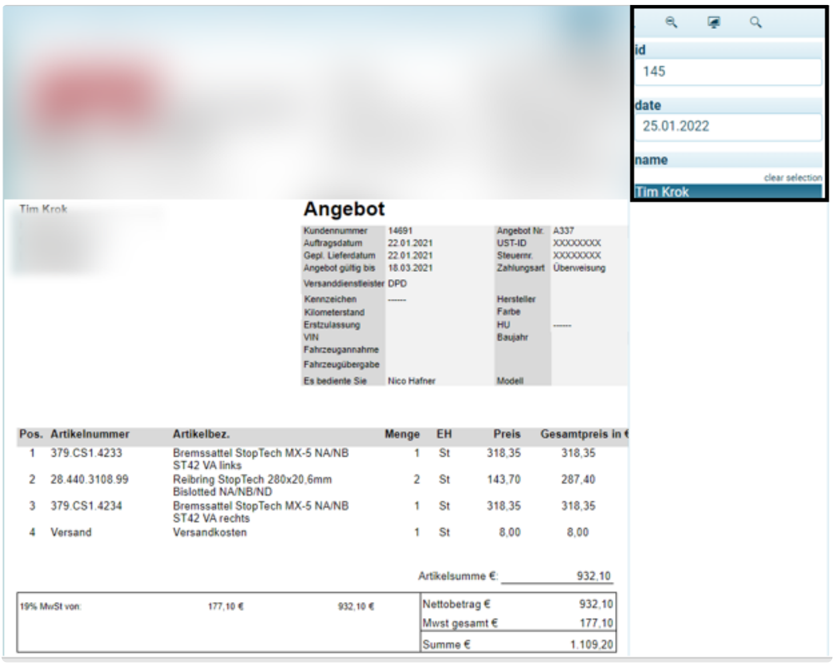
Reports

Reports

Reports are integral part of Brixxbox. They allow system user's to represent desired information to their clients. Basic example of any Point of Sale System is Order Report, it contains all the information of the user, the details of goods being purchased, and complete bill of whole order. Brixxbox allows system users to create report templates. User can create their own report templates in the form of html templates. Brixxbox keep a list of all report templates created by users and also allow their updation.

How to work with reports in Brixxbox

In this tutorial, we assume that user has a "customerofferreport". To start working with reports, click on report option under configuration in main side bar of your workspace. You will be prompted to report list page. Here you can add, remove, update, view, and download report templates. Select the play button of "customerofferreport", it will open the report at the bottom of the page. Scroll down to look at report.



id
145

date
25.01.2022

name
clear selection
Tim Krok

Tim Krok

Angebot

Kundennummer	14691	Angebot Nr.	A337
Auftragsdatum	22.01.2021	UST-ID	XXXXXXXX
Gepl. Lieferdatum	22.01.2021	Steuernr.	XXXXXXXX
Angebot gültig bis	18.03.2021	Zahlungsart	Überweisung
Versanddienstleister	DPD		
Kennzeichen	------	Hersteller	
Kilometerstand		Farbe	
Erstzulassung		HU	------
VIN		Baujahr	
Fahrzeugannahme			
Fahrzeugübergabe			
Es bediente Sie	Nico Hafner	Modell	

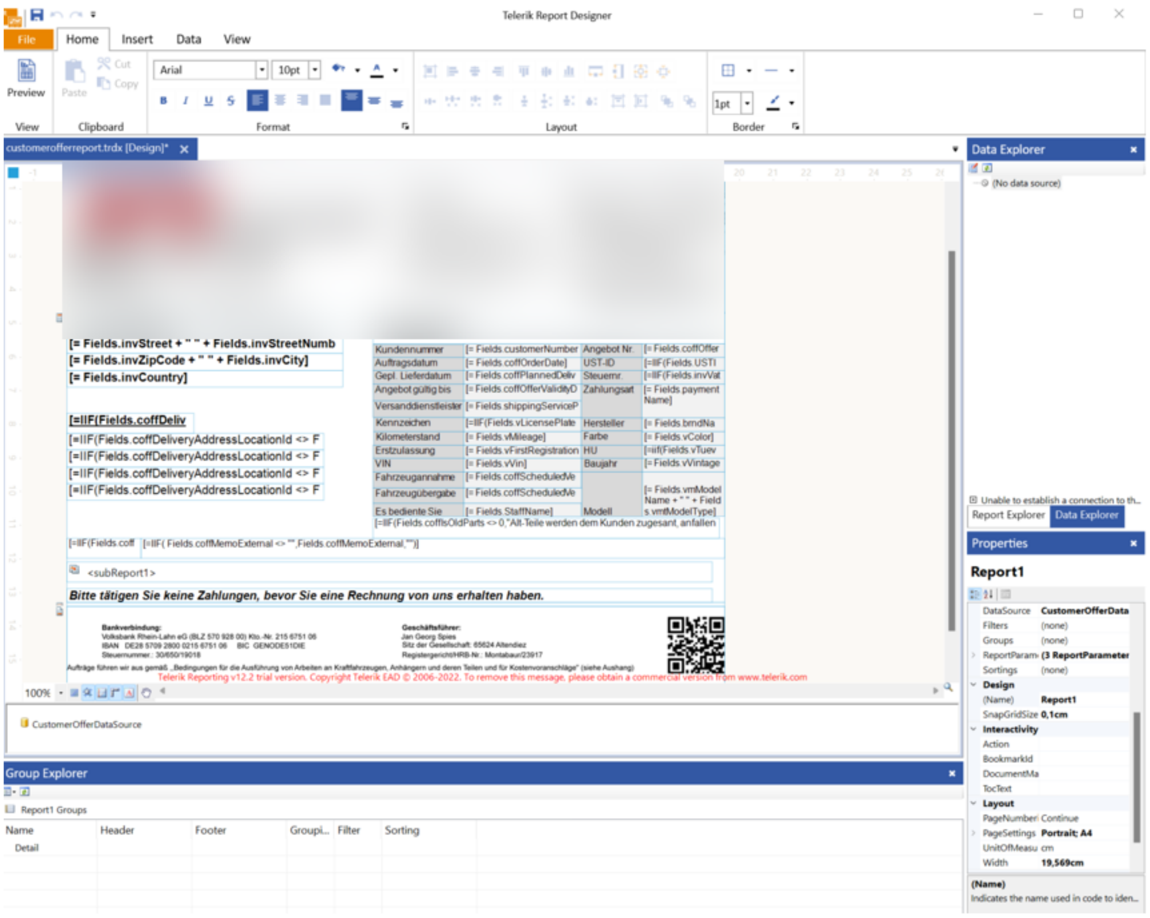
Pos.	Artikelnummer	Artikelbez.	Menge	EH	Preis	Gesamtpreis in €
1	379.CS1.4233	Bremssattel StopTech MX-5 NA/NB ST42 VA links	1	St	318,35	318,35
2	28.440.3108.99	Reibring StopTech 280x20,6mm Bislotted NA/NB/ND	2	St	143,70	287,40
3	379.CS1.4234	Bremssattel StopTech MX-5 NA/NB ST42 VA rechts	1	St	318,35	318,35
4	Versand	Versandkosten	1	St	8,00	8,00

Artikelsumme €: 932,10

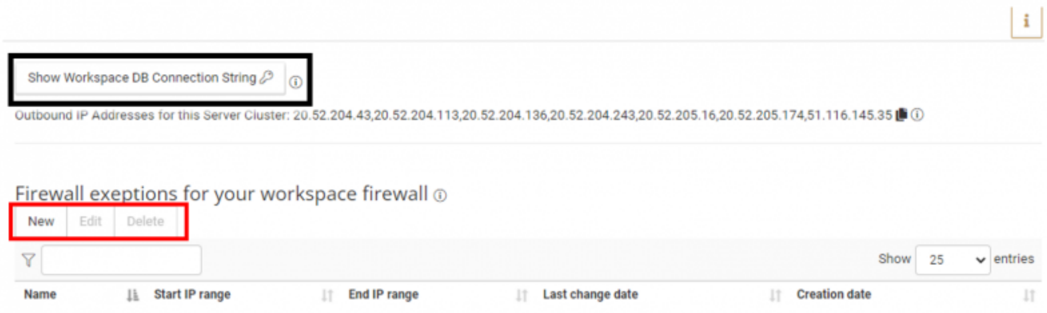
19% MwSt von:	177,10 €	932,10 €	Nettobetrag €	932,10
			Mwst gesamt €	177,10
			Summe €	1.109,20

As we can see Brixxbox allows to alter different parameters like id, date, and name (shown in black box on top left). By altering thses parameters system user can search different orders provided by the company. On the backend Brixxbox uses Telerik report designer to display or modify reports. If you want to edit or create reports Brixxbox provides you with Telerik Report Designer. You can download this software from top right corner of report's page. Lets say we want to edit customerofferreport, we need to perform follwoing steps:

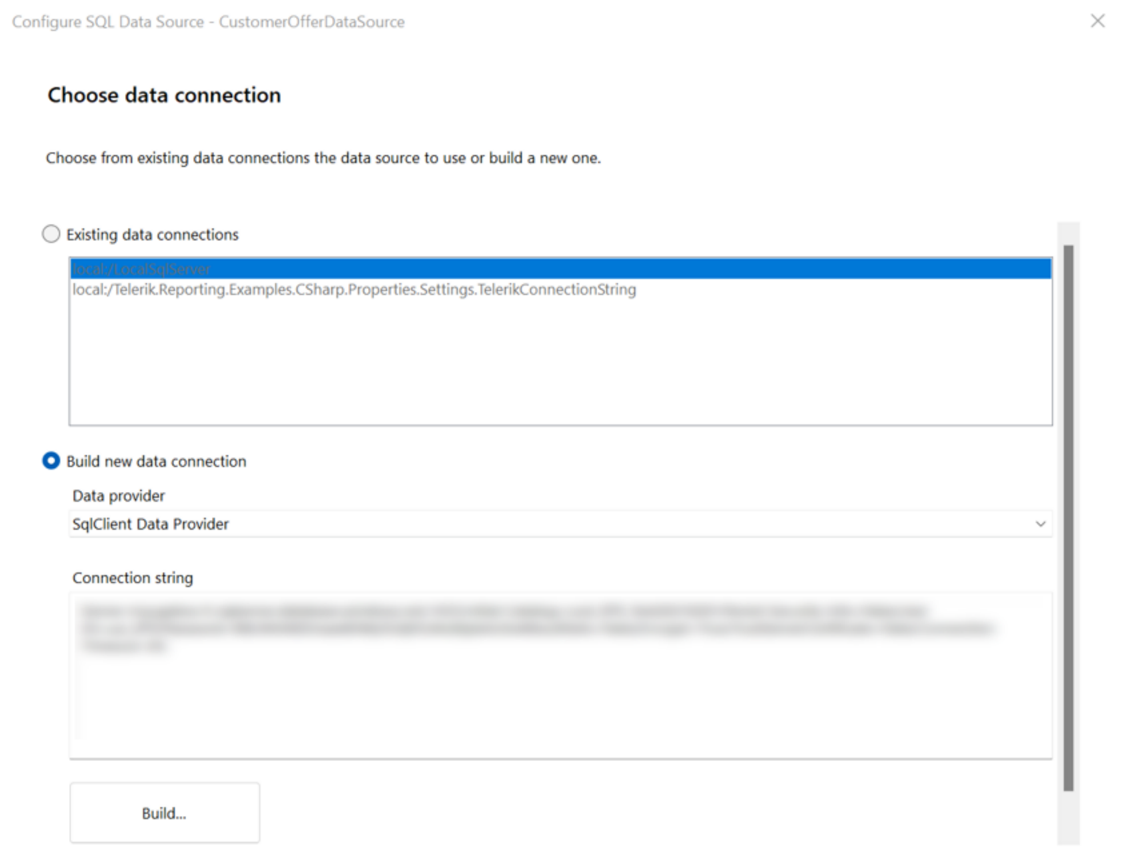
- Download "customerofferreport" from the list of reports.
- Open it in the Telerik report designer. As you can see, this is only generic template for this report and there are no specific values associated with the report like customer id. In order to add these details to see specific customer offers, we need to add a data source.



- Connect a datasource with your report. For this purpose you need to add your database connection string in Telerik. Brixxbox provides you with specific connection string. In Brixxbox main navigation panel goto Security then firewall, you can find this connection string by clicking "Show workspace DB connection string" button. In below snapshot, this button is highlighted in black. In this connection string, you can find all the important information regarding your workspace in Brixxbox like server name, password. You can also copy whole string by clicking on copy button.


Show Workspace DB Connection String
Outbound IP Addresses for this Server Cluster: 20.52.204.43,20.52.204.113,20.52.204.136,20.52.204.243,20.52.205.16,20.52.205.174,51.116.145.35
Firewall exeptions for your workspace firewall
New
Edit
Delete
Show
25
entries
Name
Start IP range
End IP range
Last change date
Creation date

- In order to protect any workspace database, all the information is made private. User need to add its ip address to firewall exception list which is also present on the same page in order to access the data. User can add its ip address by clicking on new button and then providing its public ip. User is also allowed to add a range of ip addresses. New button is highlighted in red in above snapshot.
- Now, we have DB connection string as well as a white listed user ip and we can add it in DB connection wizard of Telerik. Add it in connection string textbox and click next.


Configure SQL Data Source - CustomerOfferDataSource

Choose data connection

Choose from existing data connections the data source to use or build a new one.

Existing data connections

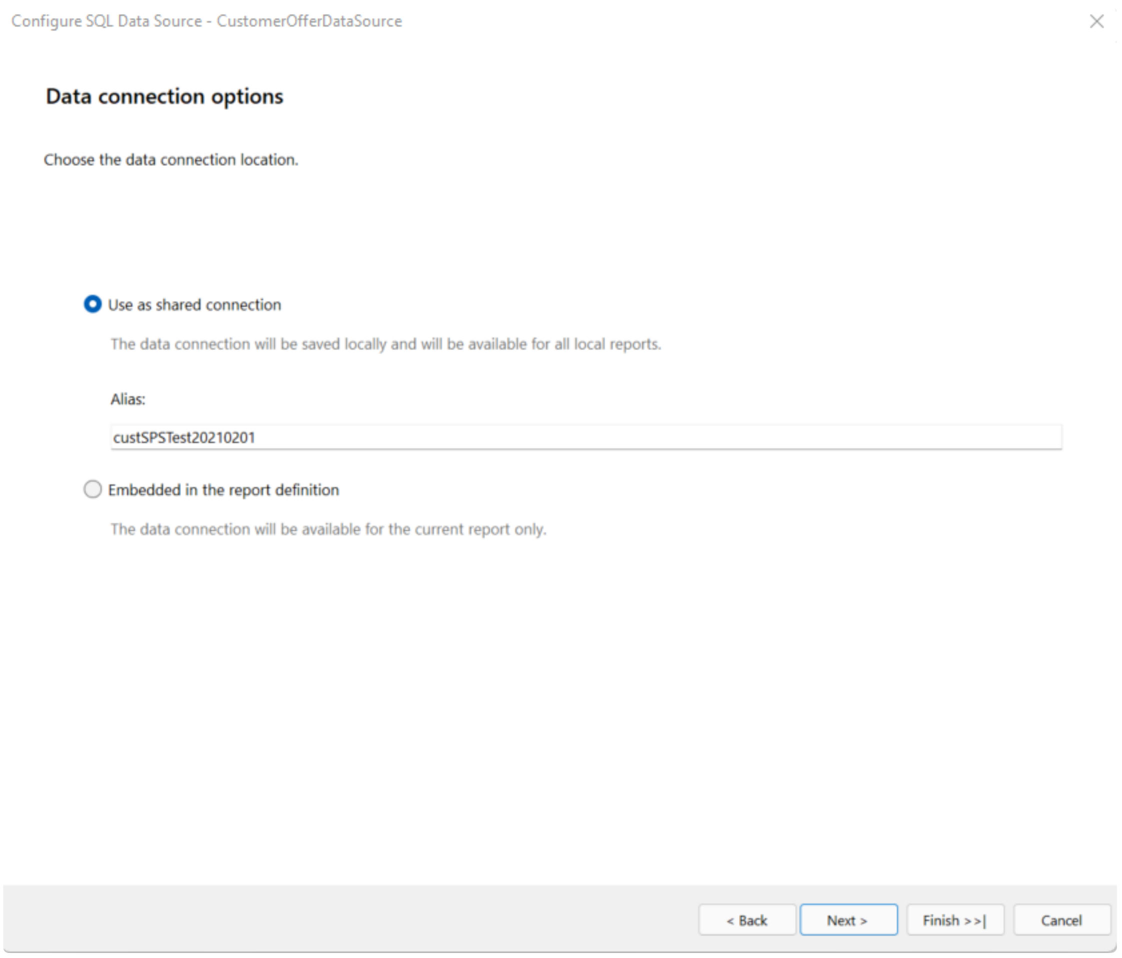
local:/LocalSqlServer
local:/Telerik.Reporting.Examples.CSharp.Properties.Settings.TelerikConnectionString

[✓ Build new data connection]

Data provider: SqlClient Data Provider

Connection string: [redacted]

- On next page, choose a shared connection so that connection will be saved locally and will be available for all local reports and give an alias name to it eg: custSPSTest20210201.


Configure SQL Data Source - CustomerOfferDataSource

Data connection options

Choose the data connection location.

[✓ Use as shared connection]

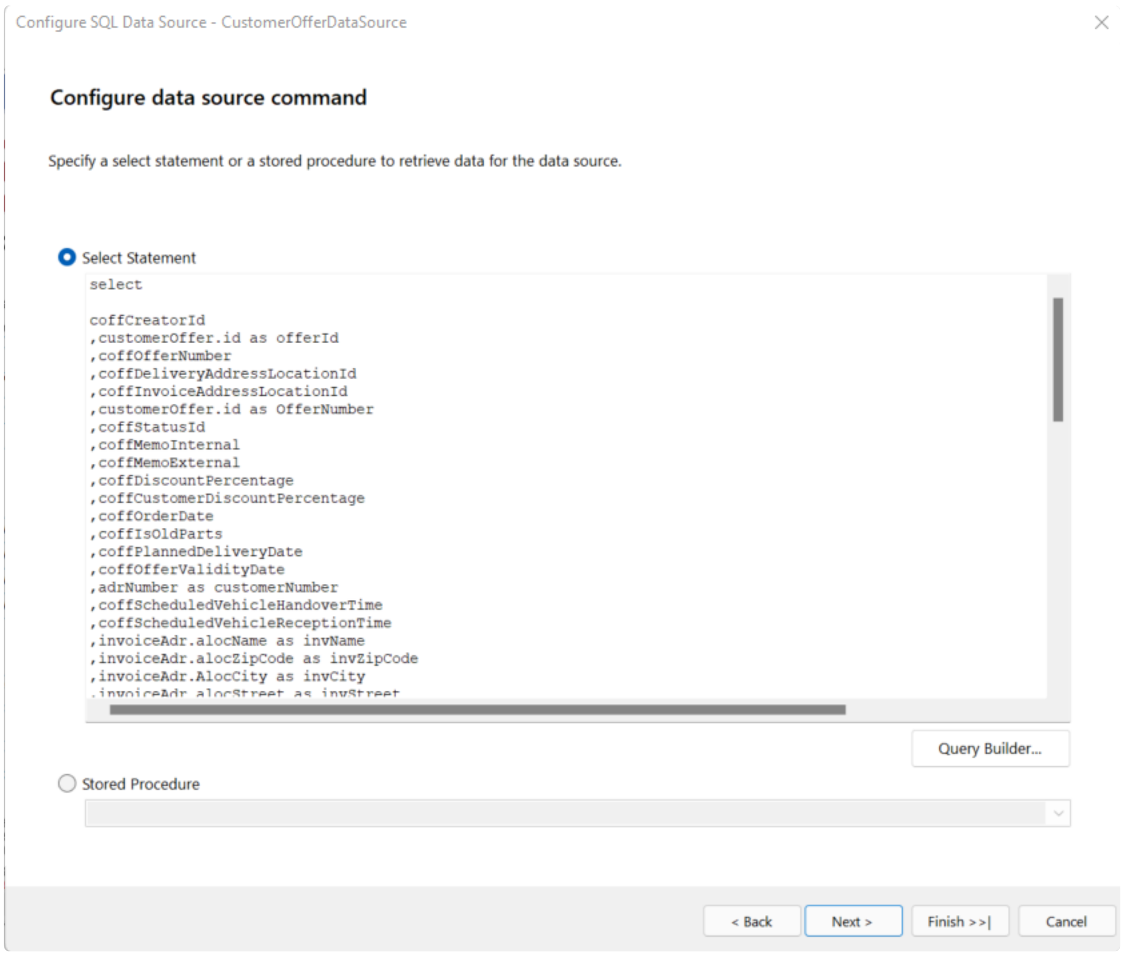
The data connection will be saved locally and will be available for all local reports.

Alias: custSPSTest20210201

Embedded in the report definition

The data connection will be available for the current report only.

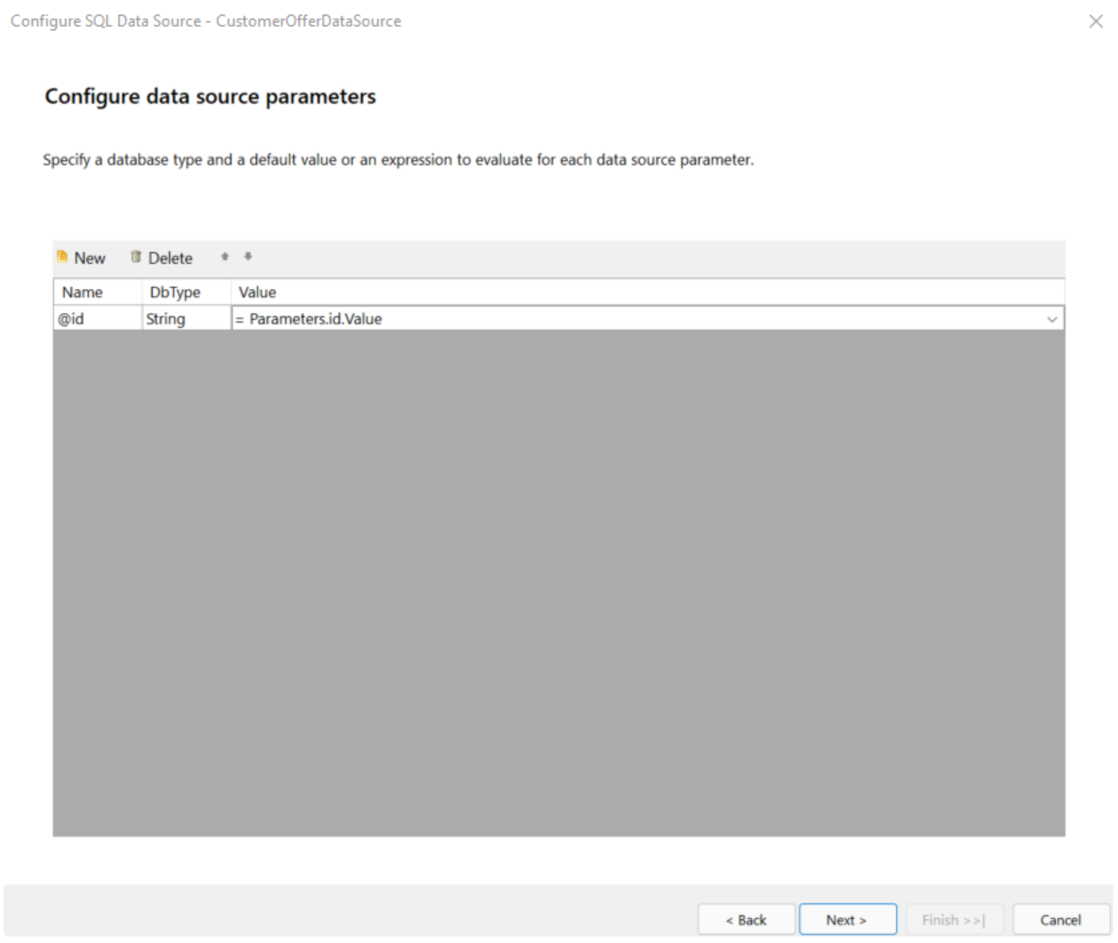
- On next page, we need to specify the sql query to get all the required data from DB regarding a valid customer offer. We can also supply a stored procedure here.



Configure SQL Data Source - CustomerOfferDataSource
Configure data source command
Specify a select statement or a stored procedure to retrieve data for the data source.
Select Statement
Query Builder...
Stored Procedure
< Back
Next >
Finish >>|
Cancel

```
select

coffCreatorId
,customerOffer.id as offerId
,coffOfferNumber
,coffDeliveryAddressLocationId
,coffInvoiceAddressLocationId
,customerOffer.id as OfferNumber
,coffStatusId
,coffMemoInternal
,coffMemoExternal
,coffDiscountPercentage
,coffCustomerDiscountPercentage
,coffOrderDate
,coffIsOldParts
,coffPlannedDeliveryDate
,coffOfferValidityDate
,adrNumber as customerNumber
,coffScheduledVehicleHandoverTime
,coffScheduledVehicleReceptionTime
,invoiceAdr.alocName as invName
,invoiceAdr.alocZipCode as invZipCode
,invoiceAdr.AlocCity as invCity
,invoiceAdr.alocStreet as invStreet
```

- On next page, we need to specify the sql statement parameters on the basis of which the query will be executed. Here we are selecting "parameter.id.Value" which is of type string.



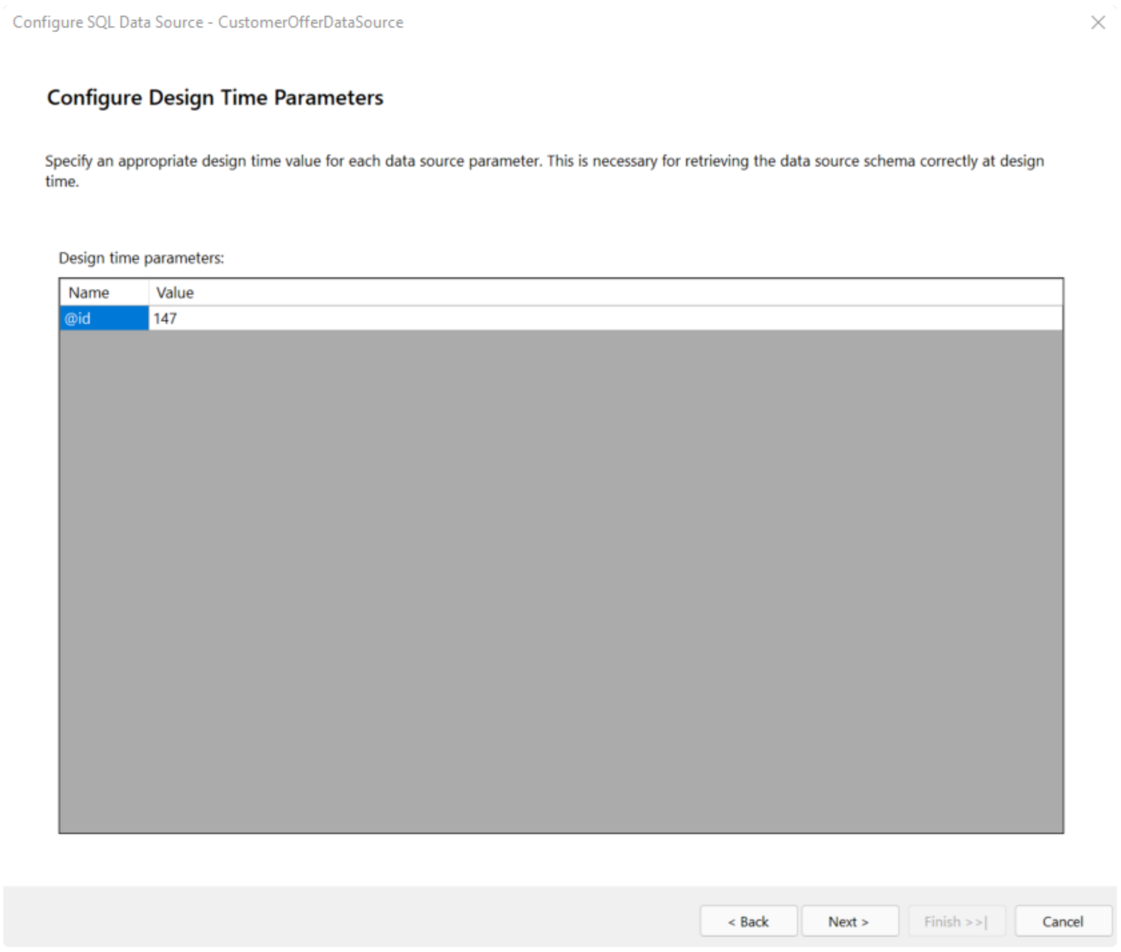
Configure SQL Data Source - CustomerOfferDataSource

Configure data source parameters

Specify a database type and a default value or an expression to evaluate for each data source parameter.

Name	DbType	Value
@id	String	= Parameters.id.Value

- In this page, pass an example value of report id i.e 147. This can be used on next page to test whether the connection with data source is successful or not. As we can see that pressing execute button, a report data with id 147 is being returned. Now user can finish the connection wizard.


Configure SQL Data Source - CustomerOfferDataSource

Configure Design Time Parameters

Specify an appropriate design time value for each data source parameter. This is necessary for retrieving the data source schema correctly at design time.

Design time parameters:

Name	Value
@id	147

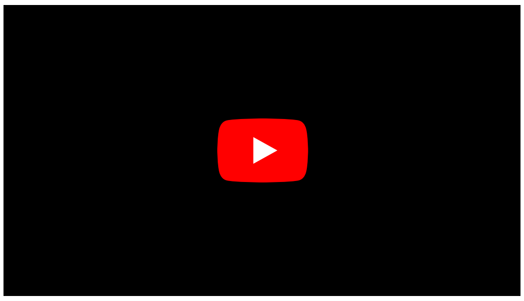
- We can also add a sub report to an existing report. Just click on sub report section and add a sub report to an existing report. There is only one limitation user needs to take care of which is, the sub report should be of type ".trdx".

Roles

Roles

This Panel allows you to create Roles for a Workspace

Demo

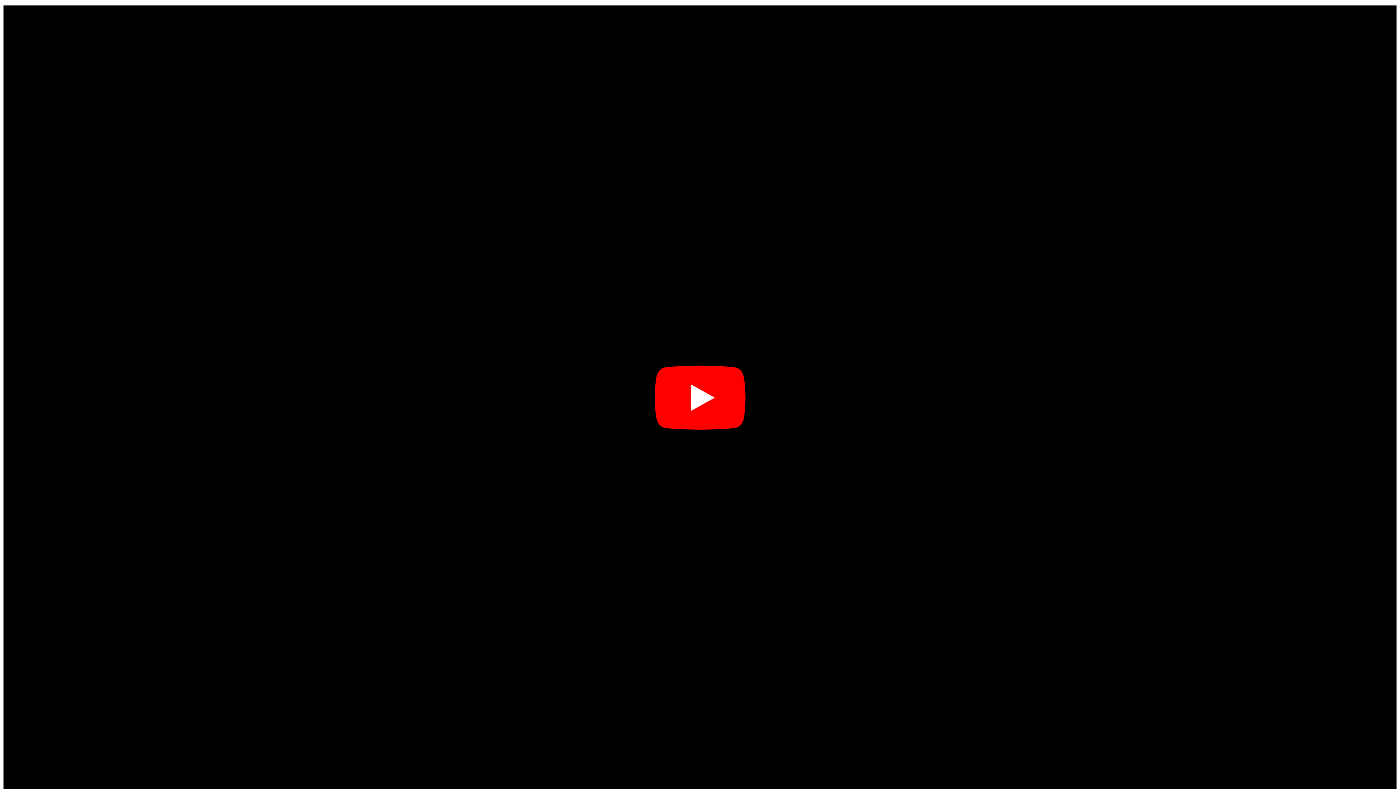


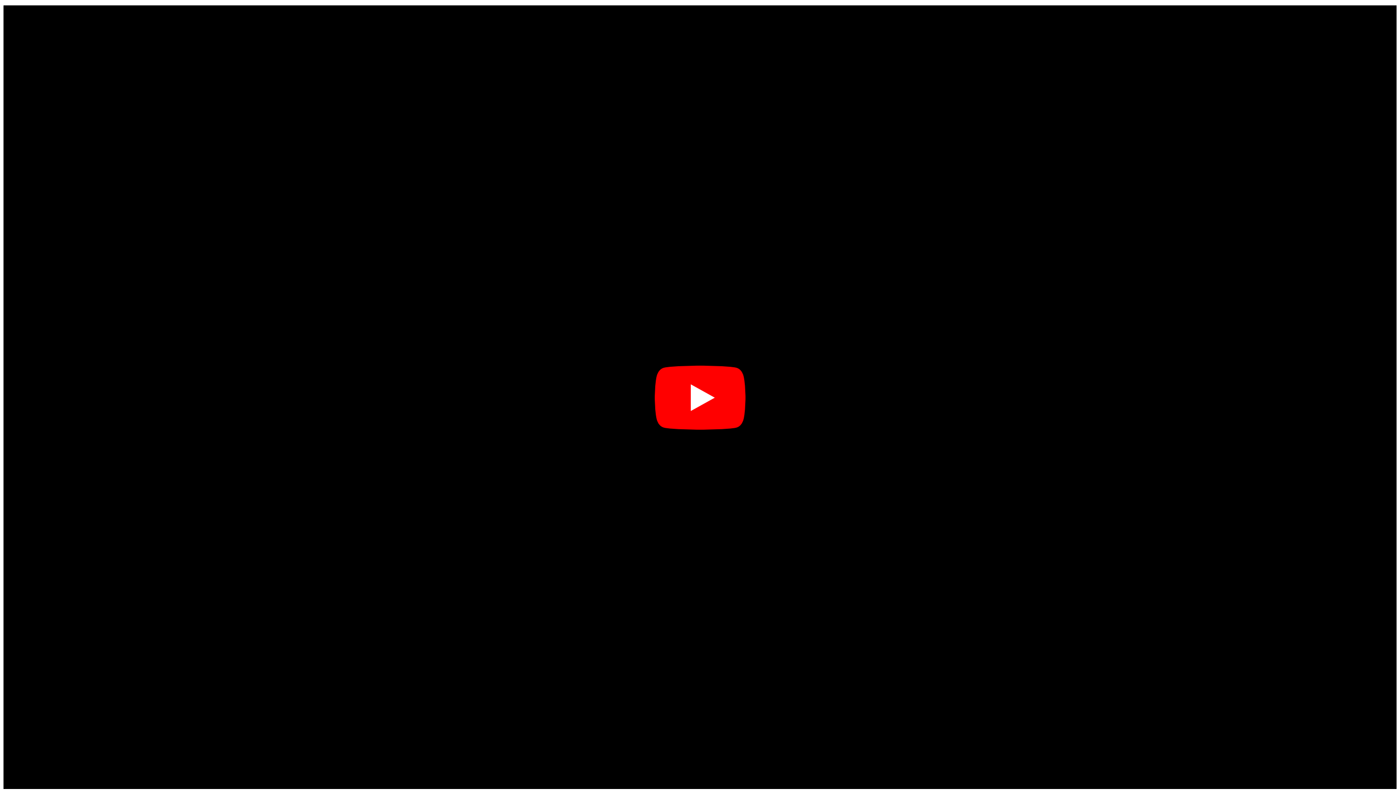
ServerSideFunction

Server Functions

Server Functions (or Server Side Functions) are functions that are executed on the server, instead of the javascript engine in the browser. Basically they are webservices, that respond to a post call and that accept the parameters in a specific pattern, defined by the brixxbox.

You can use the api function serverFunction to call a Server Function





Example Usages

To call a function and get the response object:

```
let functionResponse = brixxApi.serverFunction("myFunction", {
   name: "Hallo"
});
console.log(functionResponse.functionResult);
```

To get an example payload based on the current app state:

```
let functionResponse = brixxApi.serverFunction("myFunction", {
   name: "Hallo"
},{
   payload: true
});
console.log(functionResponse.functionResult);
```

To get an example postman collection including the payload of the current app:

```
let functionResponse = brixxApi.serverFunction("myFunction", {
   name: "Hallo"
},{
   postman: true
});
console.log(functionResponse.functionResult);
```

SqlFunction

SQL Functions

This app allows you to maintain SQL functions for your workspace DataBase.

In the main editor window you can edit the body of the function. The Create/Alter and function name must be omitted. After saving the Function it will be checked for errors. If no Errors are found it will be added to your DataBase.

It is always good practice to add a brief description for your function. This description is searchable within the function list.

Once the function is successfully added to your DataBase you can directly test it from the query window. Enter your test and execute it. Errors or results will be shown. If u enter multiple tests in the query window, select the one you want by marking the relevant text. If a selection is active, only this will be executed.

Example Usages

Only write the body of the function! Is this case the **CREATE FUNCTION ExampleDateToVeryLongString** is automatictly generated.

```
(@DATE datetime)
RETURNS nvarchar(max)
WITH EXECUTE AS CALLER
AS
BEGIN

        declare @DayAsString nvarchar(max);

    SET @DayAsString =
        CASE
            WHEN DAY(@DATE) = 1 THEN '1st'
            WHEN DAY(@DATE) = 2 THEN '2nd'
            WHEN DAY(@DATE) = 3 THEN '3rd'
                        ELSE DATENAME(day, @DATE) + 'th'
        END;

        RETURN( DATENAME(weekday, @DATE)
               + ' ' +  @DayAsString + ' of '
                   + DATENAME(month, @DATE)
                   + ' in the Year ' + DATENAME(YEAR, @DATE)
        );
END;
```

This Function can be used as part of a SQL Statement. i.g.

```
SELECT dbo.ExampleDateToVeryLongString ((GETUTCDATE()))
```

SqlStoredProcedure

SQL Stored Procedure

This app allows you to maintain SQL stored procedures for your workspace DataBase.

In the main editor window you can edit the body of the procedure. The Create/Alter and procedure name must be omitted. After saving the stored procedure will be checked for errors. If no Errors are found it will be added to your DataBase.

It is always good practice to add a brief description for your function. This description is searchable within the list.

Once the stored procedure is successfully added to your DataBase you can directly test it from the query window. Enter your test and execute it. Errors or results will be shown. If u enter multiple tests in the query window, select the one you want by marking the relevant text. If a selection is active, only this will be executed.

Example Usages

Only write the body of the procedure! Is this case the **CREATE PROCEDURE ExampleDateToVeryLongString** is automaticly generated.

```
@aReturnString nvarchar(255) output
AS
BEGIN

    DECLARE @DayAsString nvarchar(max);

    SET @DayAsString =
        CASE
            WHEN DAY(getdate()) = 1 THEN '1st'
            WHEN DAY(getdate()) = 2 THEN '2nd'
            WHEN DAY(getdate()) = 3 THEN '3rd'
            ELSE DATENAME(day, getdate()) + 'th'
        END;

        SET @aReturnString = ( DATENAME(weekday, getdate())
               + ' ' +  @DayAsString + ' of '
                   + DATENAME(month, getdate())
                   + ' in the Year ' + DATENAME(YEAR, getdate())
        );
END;
```

This can be checked in the test window.

```
DECLARE @someReturn nvarchar(255);

EXECUTE dbo.ExampleDateToVeryLongString @someReturn output;

SELECT @someReturn;
```

SqlTrigger

SQL Trigger

This app allows you to maintain DML triggers for your workspace dataBase tables.

In the main editor window you can edit the body of the trigger. The Create/Alter ,trigger name and table name must be omitted. After saving the trigger it will be checked for errors. If no Errors are found it will be added to your DataBase.

It is always good practice to add a brief description for your trigger. The description is searchable within the trigger list.

Example Usages

Only write the body of the trigger! Is this case the **CREATE Trigger ExampleTrigger ON ExampleTable** is automaticly generated.

```
after UPDATE
AS RAISERROR ('Update on ExampleTable', 15, 1);
```

StandardDatasources

We are writing documentation. This page will be updated soon!

StandardMessages

Custom messages

User can add thank you, warning, and error messages here and they can be used afterwards anywhere in your app. Brixxbox allows custom messag
placeholders that can be queried and translated by script. These test parameters are captured in Json format.

Example Usages

Here we will consider a thank you message. We want to display this message after successful placement of order by the customer on our plateform.
message more personalized we also want to show the customer name and order id in the thank you message. To generate a custom message follow
below:

- From side panel of your work space click on configuration then on "custom messages". It will open custom messages app.
- Click on create, a pop up window will be opened. Add the name, description of custom message. Also add test parameters in json format.

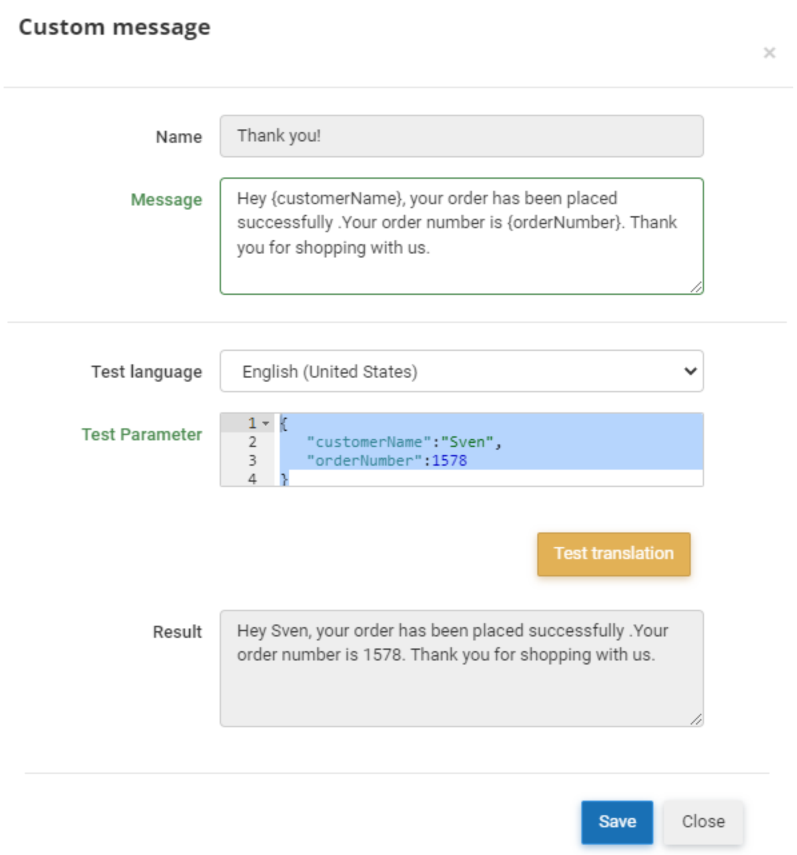
```
Name : Thank you!
```

```
Message: Hey {customerName}, your order has been placed successfully .Your order number is {orderNumber}. Thank you for shopping
```

Test parameter for the above sentence

```
{
   "customerName":"Sven",
   "orderNumber":1578
}
```

The edit panel should look like this:



Custom message

Name: Thank you!

Message: Hey {customerName}, your order has been placed successfully .Your order number is {orderNumber}. Thank you for shopping with us.

Test language: English (United States)

Test Parameter:

```
{
   "customerName":"Sven",
   "orderNumber":1578
}
```

Result: Hey Sven, your order has been placed successfully .Your order number is 1578. Thank you for shopping with us.

Now click on save button and your custom message is stored permanently in this app.

StdSqlEditor

SQL Statements

This app allows you to store SQL statements that can be used in all the apps of a workspace.

In the main editor window you can edit the SQL statement. It is always good practice to add a brief description for your statement. This description is searchable within the statement list.

In addition test parameter can be supplied. Test parameter must use Json format.

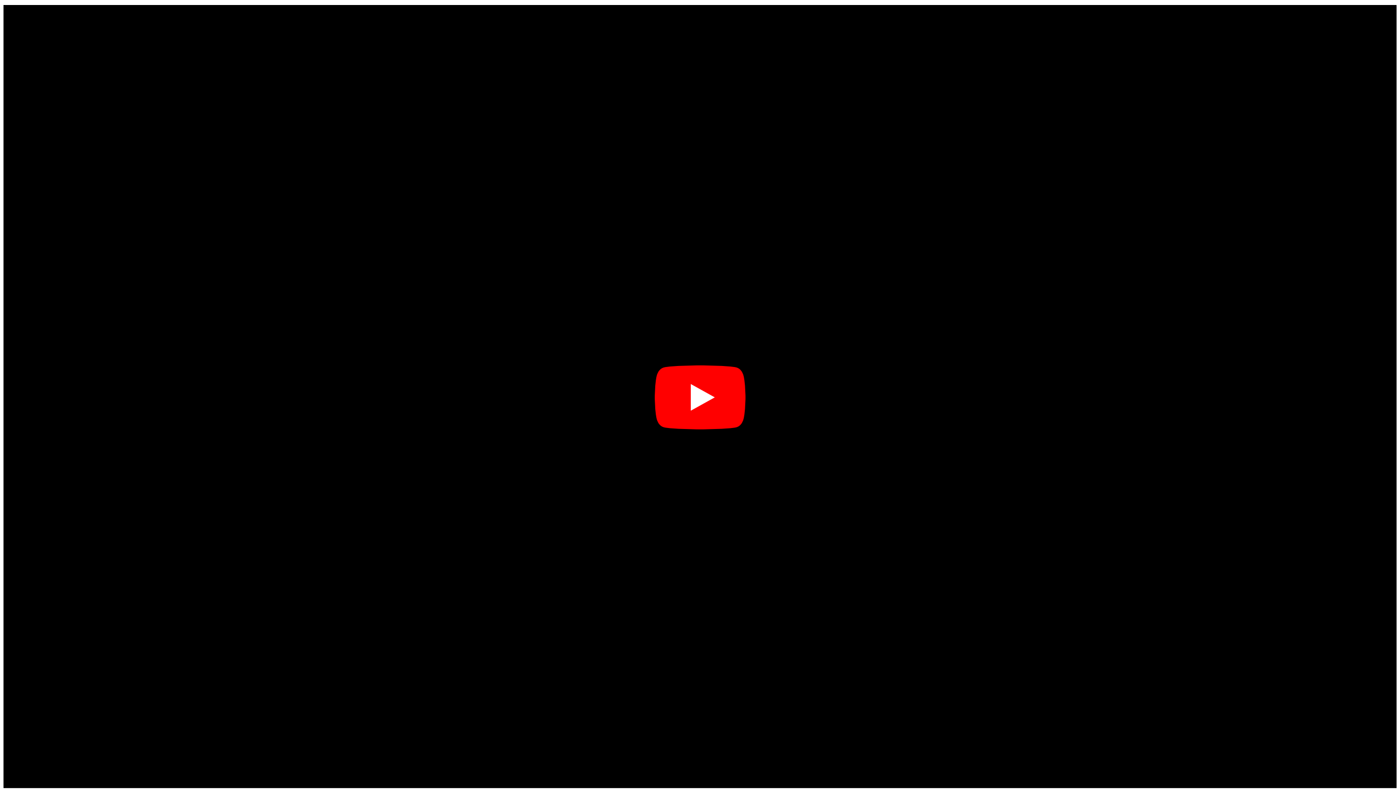
Example Usages

```sql
select * from ExampleDataBase where ExampleCountryCode = @country and ExampleZipCode = @zip
```

Test parameter for the above statement

```json
{"zip":"12345", "country":"DE"}
```

Demo



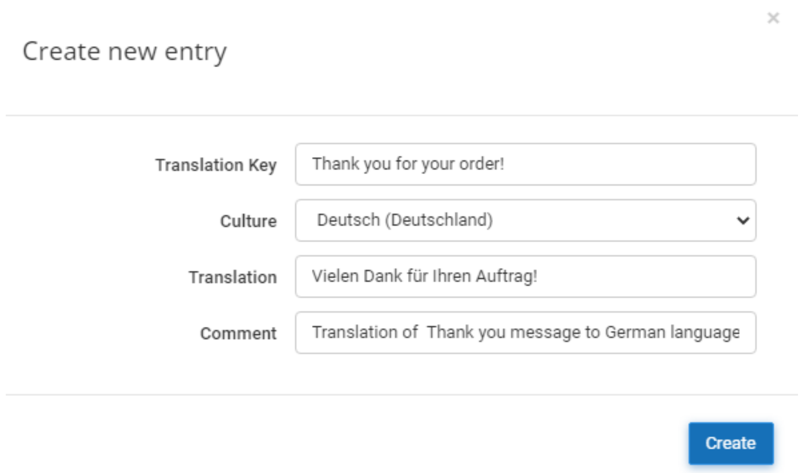
Translations

App Translation

Brixxbox allows more than 16 languages to be used in the workspace. App translation helps user to handle language conversion. This app allows user to add different phrases, messages and their translations. These messages get stored in a list format and can be used later anywhere in your workspace. Each translation app enrty consists of four properties: Translation key which is the actual message or phrase that needs to be translated, culture it is the language in which we want to translate our message, translation it is the actual translation, and last one is comment about this translation.

Example

Lets now look at one simple example a "Thank you" message after placing an order. We want this English message as well as its translation in German. For this purpose, we will a new entry in app translation. Go to main panel of your workspace then configurations then click on translations. Now in this app you will see a list of already present translations. Click on new button to add a new entry. Fill all necessary information and after this our entry should look like this.



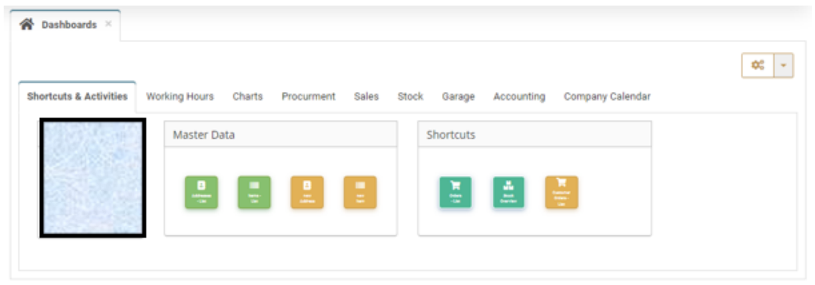
Translation Key	Thank you for your order!
Culture	Deutsch (Deutschland)
Translation	Vielen Dank für Ihren Auftrag!
Comment	Translation of Thank you message to German language

Now create this translation entry by clicking on create button.

UserDashboards

User Dashboards

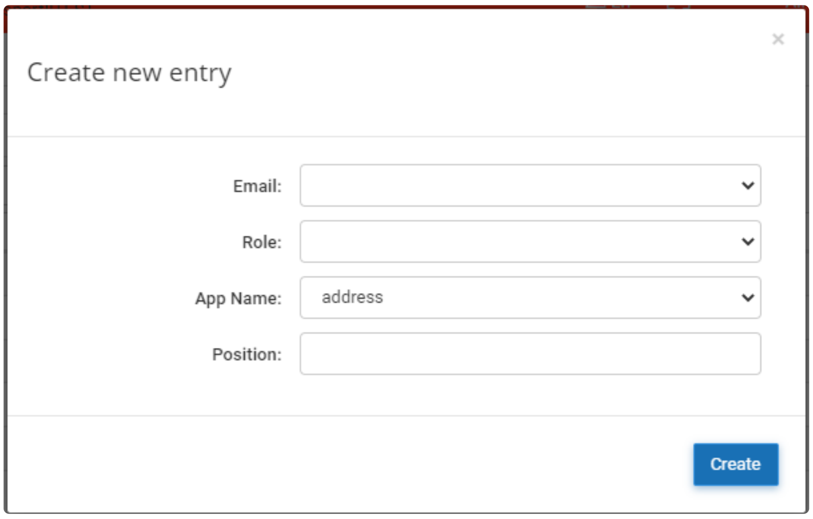
User dashboard is a whole panel or a page dedicated to creating shortcuts for users workspace. Here Brixxbox allows users to add apps which are the most important ones. Each of the app will be shown on the dashboard page. User can access this page by clicking on the logo of their workspace. Here is an example workspace.



To add different apps to this dashboard section, user should follow these steps. From left navigation panel of Brixxbox goto configuration then scroll down and select "user dashboards". Here you will find a list of all the apps that are available on dashboard panel and Brixxbox also allows system users to manage access of different dashboards for different users. An example list of dashboards for different users is given below.

In this snapshot we can see that for multiple user, we are selecting same app "dashboardDemo". We can also choose different apps here.

To add a new app in a dashboard for specific user group, select new on the users dashboard page. A new panel will be opened. It will look like this.



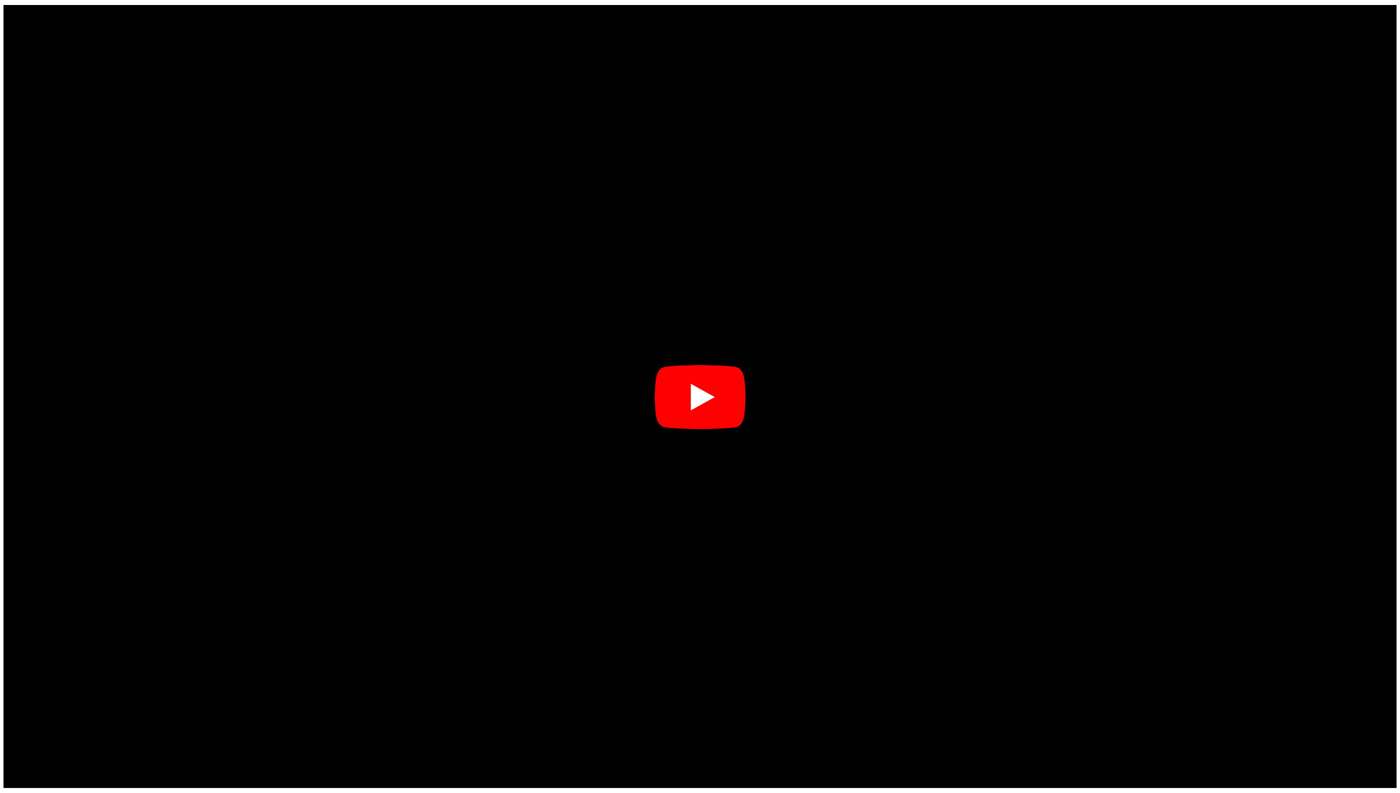
Create new entry

Email:	
Role:	
App Name:	address
Position:	

Create

Select the respective properties and click create button to add a new app to specific users dashboards in Brixxbox. In the tutorial below, user can find how to add a new app to dashboard.

Demo



Users

Users

This Panel lets you invite users to your workspace, assign roles to them and kick them eventually.

Demo

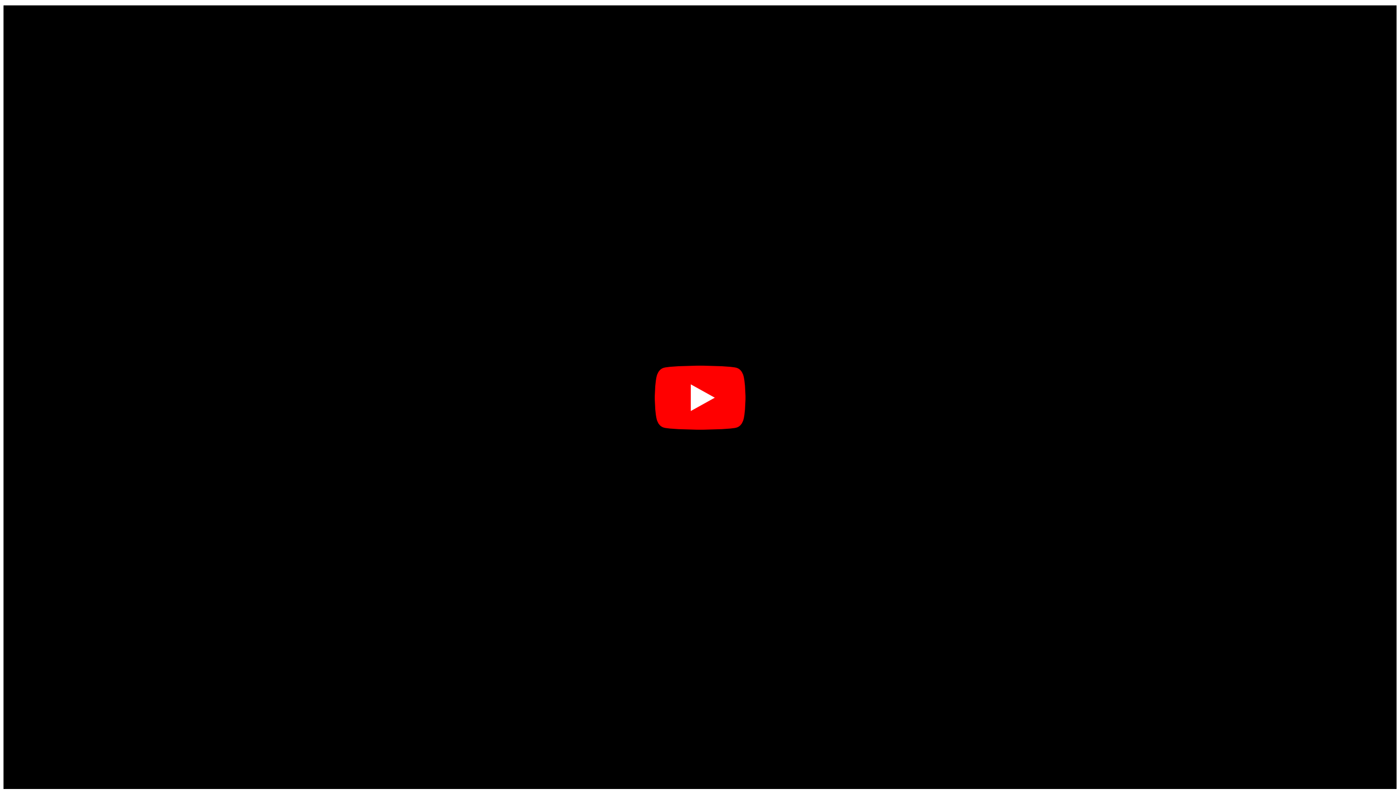
WorkspaceFirewall

Database Firewall

This app allows you to open the firewall of your brixxbox database for specific client pc. This is useful, if you need to connect directly to your database via Azure Data Studio or any other client software.

The database is a Microsoft Azure SQL Database, in case your client needs specific information for the connection type.

Demo



WorkspaceSettings

Workspace Settings

This App allows to define settings for different scenarios.

CustomInviteEmailContent

CustomInviteEmailContent

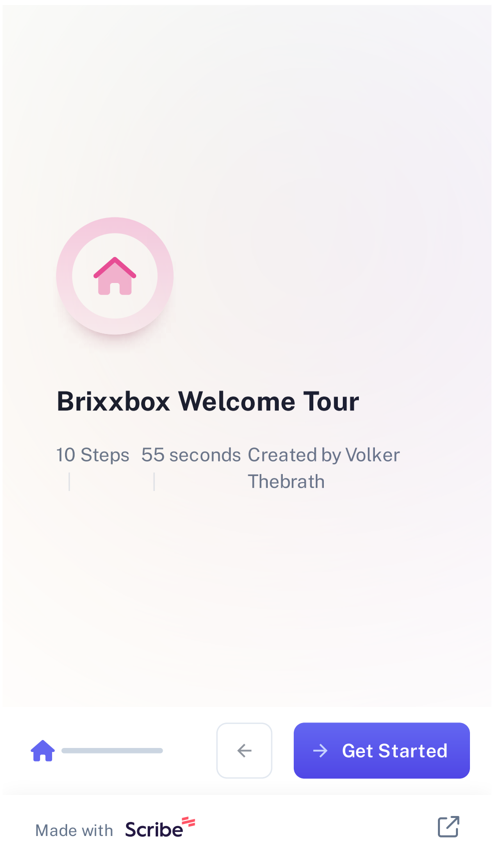
You can set the html email body for the invitation emails to introduce them to your workspace. You can use the placeholder [Link] if you want to specify the link location. If no [Link] Tag is found in your body, the tag will be appended at the end.

Attachments like the workspace custom logo can be added by using its content id: 'cid:CustomLogo.png'

```
<img src="cid:CustomLogo.png" />
<h1>Welcome to our Workspace</h1>
Please click this [Link] to choose your password.
```

How Tos

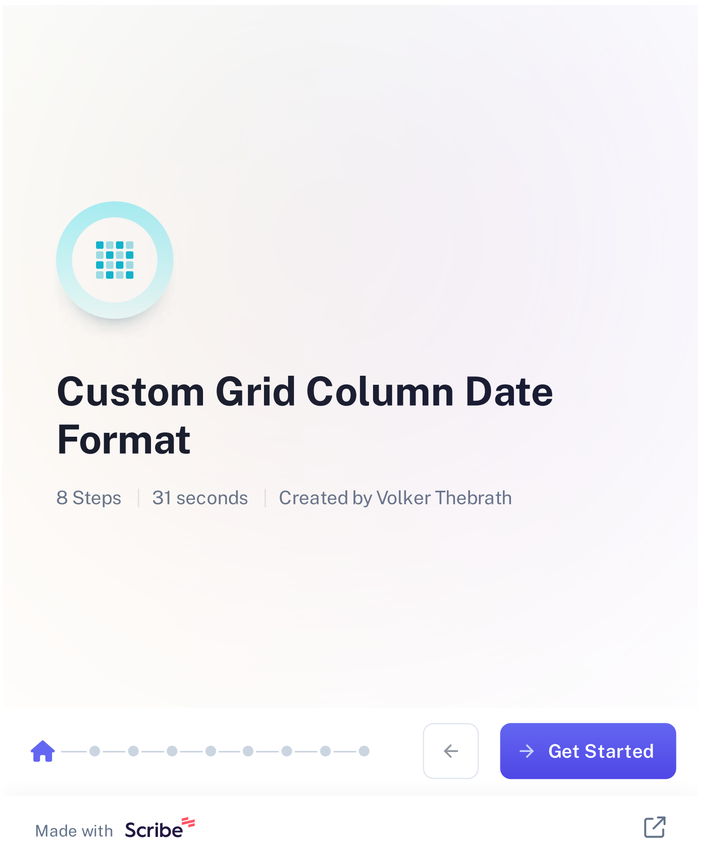
Brixxbox Welcome Tour



Brixxbox Welcome Tour

10 Steps | 55 seconds | Created by Volker Thebrath

Custom Grid Column Date Format



Custom Grid Column Date Format

8 Steps | 31 seconds | Created by Volker Thebrath

Create Report REST

Example of a HTTP Request to the brixxbox

Create Report REST

You can create a report from 3rd party systems by using a http POST call to the brixxbox.

We created a postman solution to demonstrate the call because you have to login first. Run in Postman

3 Environment Variables have to be set in Postman: "UserEmail" and "UserPassword" for the login call an "Workspace" for the url part of the CreatePdf call

You can specify if the report should be archived or not in the post parameters:

```
// POST call to https://app.brixxbox.net/w/{{Workspace}}/c/default/reporting/CreatePdf
{
   "reportName": "addresslist",

   //optional parameters
   "archive": false, //optional, false is default
   "configName": "address", //optional, only if you want to archive
   "documentTypeId": 1, //optional, only valid if configName and parameter 'id' is set and archive is true
   "culture": "de-DE", //optional, report has its defaults
   "parameters": { //optional
       "id": 1
   }
}
```

Telerik Extension Functions

Telerik Page Footer Sum

How to use brixxbox custom functions in telerik

BrixxPageFooterSum

Sums elements in a table up to this footer

Each page sum function can only be used once in a report. Using it multiple times will result in wrong calculated values!

Demo

```
//This function belongs in a PageFooterSection of a telerik report
=PageExec("textBoxValue",            //put in a textbox from your table row. Typically your value textbox, but whatever is in yo
           BrixxPageFooterSum(
               Fields.yourValue,      //put in the Value Field you want to sum up
               ReportDefinition))     //this is fix (it will tell the sum to reset for each new report)
```

Telerik Page Header Sum

Sums elements in a table up to this header (carry over from previous footer sum)

Each page sum function can only be used once in a report. Using it multiple times will result in wrong calculated values!

```
//This function belongs in a PageHeaderSection of a telerik report
=PageExec("textBoxValue",            //put in a textbox from your table row. Typically your value textbox, but whatever is in yo
           BrixxPageHeaderSum(
               Fields.yourValue,      //put in the Value Field you want to sum up
               PageNumber,            //this is fix (the PageNumber variable from your report. this will tell the function when t
               ReportDefinition))     //this is fix (it will tell the sum to reset for each new report)
```

Printing in LAN

How to print on a local Printer in your LAN