Creatio Academy

Mobile development

Mobile application basics

Version 8.0

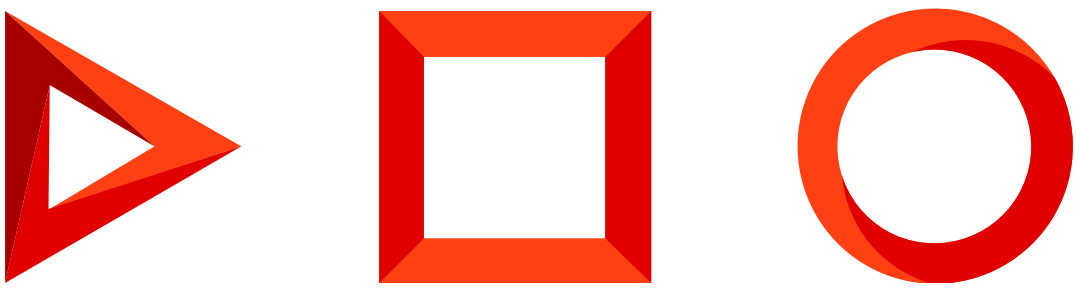


Table of Contents

Mobile application basics

Easy

Mobile app types

Mobile app type is based on its implementation type.

View the mobile app types in the table below.

Mobile app types

Type	Description
Native mobile app	An app developed for a particular mobile platform (iOS, Android). Installed from the app store. Such an app is developed using a high-level language and compiled into native OS code to ensure the best performance. It cannot be transferred among mobile platforms easily.
Mobile web app	A specialized website adapted to specific mobile devices. The app is platform independent. This requires a constant network connection, as the app is physically hosted not on the mobile device but on a dedicated server.
Hybrid app	A mobile app wrapped in a native container. Installed from the app store. Such app is developed using HTML5, CSS and JavaScript languages. These apps are easy to transfer among mobile platforms, but have slightly lower performance compared to native apps.

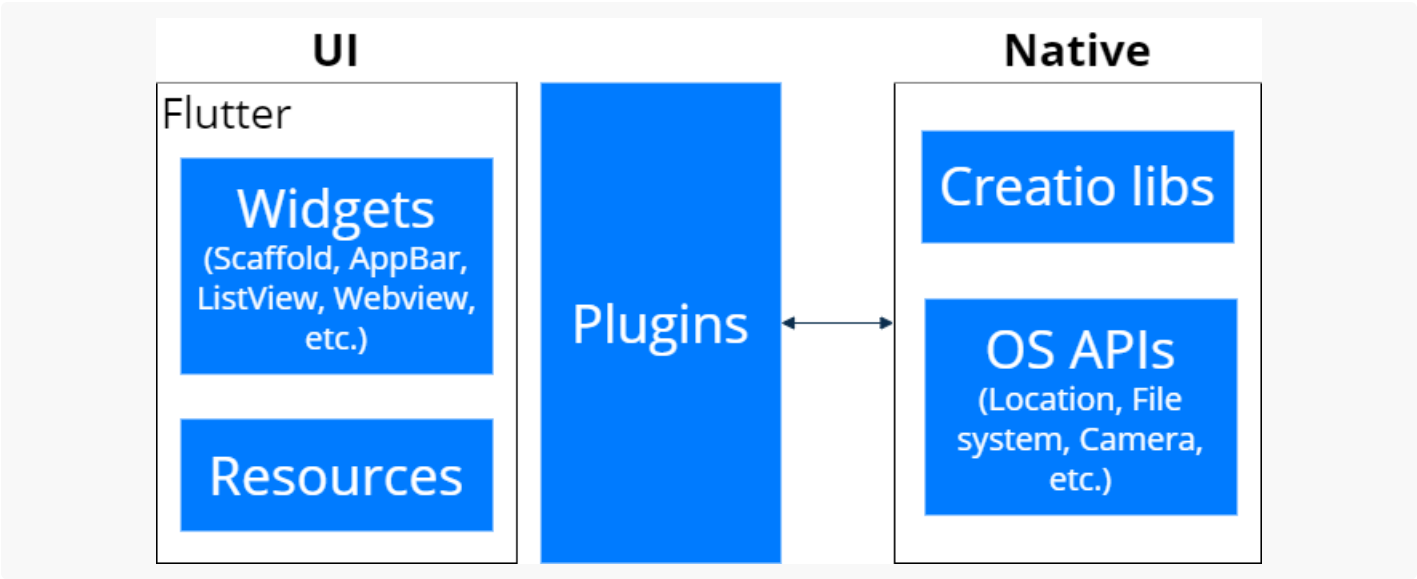
Creatio mobile app architecture

The **Creatio mobile app** is a remote workplace that provides quick access to customer data, calendar, mobile mailing list, etc.

The Creatio mobile app is available for download on the App Store and Google Play on mobile devices that meet the requirements. Learn more: System requirements for mobile devices (user documentation).

Implementation

View the mobile app architecture chart in the figure below.



The main framework of the Creatio mobile app is Flutter. Learn more: Wikipedia.

Flutter framework **features**:

- Performance of apps developed using this framework is close to native apps (native-like).
- The mobile apps developed using this framework have the advantages of hybrid apps when it comes to using common code for iOS and Android mobile platforms.

Flutter framework **benefits**:

- A large number of dashboards to build the UI.
- Access to the mobile device programming UI (API) that interacts with the database, file system, mobile device sensors. For example, camera.

Learn more: official vendor documentation.

The core of the Creatio mobile app is a **set of libraries** (Creatio libs). The connection between Flutter and the mobile app core is implemented using plugins developed in Java (Android) or Objective C (iOS). The main part of the library code is implemented in Java and is cross-platform. Cross-platform functionality is ensured using the J2ObjC utility that converts Java code into Objective C code. Learn more: official vendor documentation.

Libraries let you execute the following actions:

- Perform synchronization by communicating with the Creatio server.
- Work with the file system.
- Work with metadata received upon synchronization, etc.

Metadata are configuration files the app retrieves as part of synchronization with the Creatio server and stores locally in the device file system.

Metadata includes:

- Creatio mobile app manifest.
- Schemas that extend the app capabilities.
- Settings of sections that the Mobile App Wizard creates.

Note. The app also uses the Apache Cordova framework during the functionality transition to the Flutter framework. Learn more: Wikipedia.

Workflow

The store-published Creatio mobile app published is a set of modules needed to synchronize with the desktop Creatio application server. The desktop Creatio application stores all settings and data needed for the mobile app. View the mobile app workflow in the figure below.

After you install the app into the mobile device and set the connection parameters to the Creatio server, the app receives metadata (app structure, system data) and data from the server.

The advantage of this workflow is compatibility with all existing Creatio products. Each product and each customer website can contain its own set of mobile app settings, business logic, and even UI. All a mobile user needs to do is install the mobile app and synchronize it with the Creatio website.

Operation modes

Operation modes of mobile apps

Mode	Description
Hybrid mode	Hybrid mode is designed for working with data and is activated automatically if a stable connection to the Creatio server is unavailable. This mode lets you create new records and work with schedules. It is also possible to manage 10 section records with which you have interacted recently.
Online	Online mode requires an Internet connection. In this mode, the user works directly with the main application that works as Creatio server. Configuration changes are synchronized automatically in real-time.
Offline	For offline mode, an Internet connection is only required for the preliminary import and synchronization. When you use this mode, the app saves data locally to the mobile device. You must synchronize data with the Creatio server manually to receive configuration changes and data updates.

To change the operation mode **for a single user**:

1. **Open the System settings section**. To do this, click → **System setup** → **System settings**.
2. **Open the Mobile application operation mode** (`MobileApplicationMode code`) system setting.
3. **Fill out the system setting properties**.

Property	Property value
Default value	Online or offline
Save value for current user	Select the checkbox

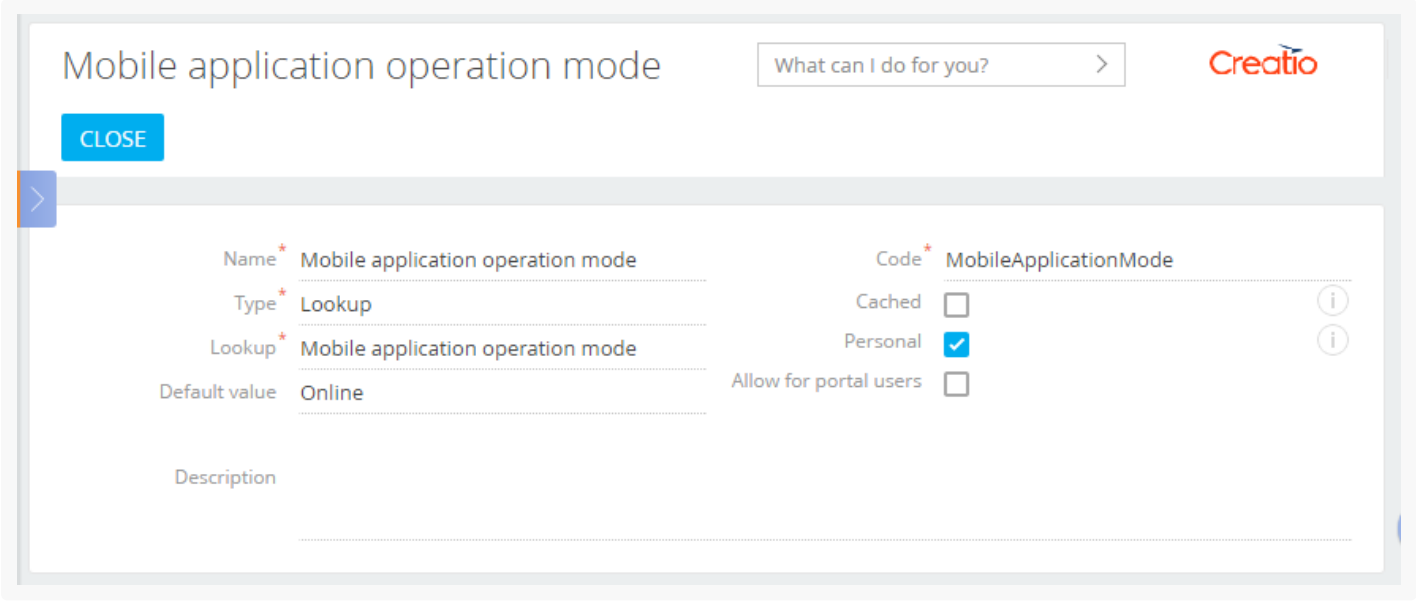
4. **Save the changes**.

To change the operation mode **for all users**:

1. **Open the System settings section**. To do this, click → **System setup** → **System settings**.
2. **Open the Mobile application operation mode** (`MobileApplicationMode code`) system setting.
3. **Fill out the system setting properties**.

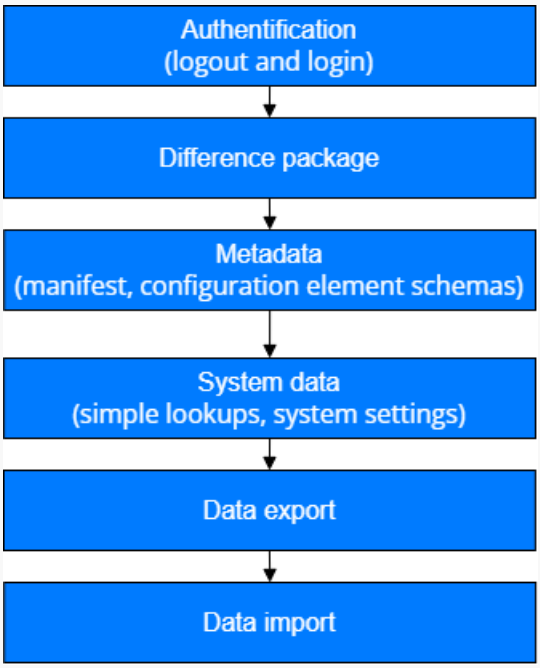
Property	Property value
Default value	Online or offline
Save value for current user	Clear the checkbox

4. **Save the changes**.
5. Make sure that users **have permission to change the system setting**.



Synchronization

Synchronization with the Creatio server performs different tasks based on app operation mode. In online mode, synchronization is only needed to receive configuration changes. In offline mode, synchronization is needed to receive updates and send / receive changed or new data. View the general synchronization workflow below.



First, the app performs authentication. The current active session is destroyed on the server. Next, the app requests data from the server to generate a difference. The app analyzes this data and requests modified or new configuration schemas. After the schemas are loaded, the app retrieves system data, which includes cached lookups (also known as simple lookups), system settings, etc. Then, data is exchanged with the server.

The mobile app implements one more synchronization stage called “Actualize data.” If this functionality is enabled, it is the last synchronization step. The app compares data available on the server with local data. If a difference is found, the app downloads missing data or deletes outdated data. This mechanism resolves issues that can occur when redistributing access permissions or deleting data from the server. To **enable data actualization**:

1. **Open the mobile app manifest.**.
2. **Go to the** `SyncOptions` **section →** `ModelDataImportConfig`.
3. **Set the** `IsAdministratedByRights` **property** to `true`.

Export data in batch mode

By default, the mobile app sends each change to data one by one. Thus, one change makes at least one request to the server. If the number of changes is large, executing such requests can take time.

The app lets you send data in batch mode, which accelerates data upload to the server significantly.

To **enable the batch mode of data sending**:

1. **Open the mobile app manifest**.
2. **Move to the** `SyncOptions` **section**.

3. **Set the** `UseBatchExport` **property** to `true`.

As a result, all user changes will be grouped into multiple batch requests by operation type. The available operation types are insert, update, and delete.

Life cycle of mobile app pages

When you navigate through the mobile app, Creatio executes multiple steps for each page. The **stages** in the life cycle of mobile app pages are as follows:

- Open. Read more >>>
- Close. Read more >>>
- Upload. Read more >>>
- Return to the page. Read more >>>

The time from loading a page into the memory of a mobile device to its final unload from memory is called the page life cycle. **Page events** are provided for each life cycle stage. The events let you extend the functionality. The base events are as follows:

- initialize view
- finish class initialization
- load a page
- load a data
- close a page

Since a phone screen can display only a single page and a tablet screen can display a single page in portrait mode and two in landscape mode, the page life cycle for phone and tablet is different.

Open a page

When you open a page the first time, the mobile app loads the scripts needed for the page work. Next, the app initializes the controller and creates the view.

View the generation sequence of page open events in the table below.

Event	Description
initializeView	Initializes views
pageLoadComplete	Completes page loading
launch	Initiates data loading

Close a page

While closing a page, Creatio removes its view from the object model and removes the controller from device memory.

Creatio closes the page in the following cases:

- When you click the [*Back*] button. In this case, the last page is deleted.
- When you move to another section. In this case, previously opened pages are deleted.

`pageUnloadComplete` event finishes page closure.

Unload a page

Creatio unloads a page when you move to another page in the same section. The current page becomes inactive. It can remain visible on the device screen. For example, if you open a page from the list on a tablet, the list page remains visible. In the same case on the phone, the list page becomes invisible but remains in memory. There is a difference between unloading the page and closing it.

`pageUnloadComplete` event (the same as the page close event) unloads the page.

Return to a page

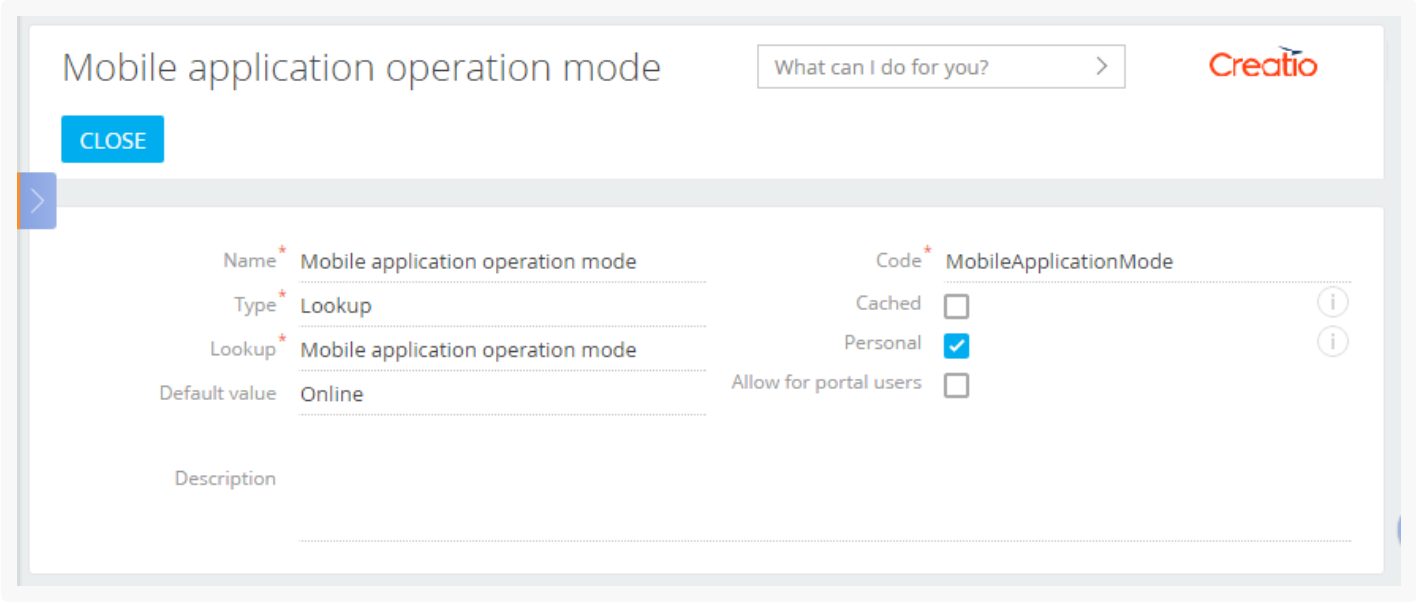
The app returns to the previously unloaded page when you click the [*Back*] button.

`pageLoadComplete` event returns to the page.

Attention. The app can use only one page instance. Therefore, if you open two identical pages one by one, the `launch` event handler is executed again when you return to the first page. Keep it in mind during development.

Background configuration update in a mobile app

Creatio mobile app implements a synchronization mechanism for the app structure that can work automatically in the background. To manage this process, use the [*Update checks frequency*] (`MobileAppCheckUpdatePeriod` code) system setting.



This setting sets the time in hours after which the mobile app can request configuration changes from Creatio. If set to 0, the app always downloads configuration updates when they appear.

Conditions to run the synchronization

The conditions to run the structure synchronization in the background are as follows:

- The mobile device uses the iOS or Android platform.
- The synchronization was not run previously.
- More time has passed since the structure was last synchronized than the time specified in the [*Update checks frequency*] (`MobileAppCheckUpdatePeriod` code) system setting.
- The device launches or activates the app, i.e., the app was previously minimized or you access it from another app.

If changes are received during the structure update, the app is restarted automatically to apply them when you minimize the app or move to another app.

Special features on different platforms

1. Background mode on the **Android** platform is implemented using a parallel running service. This approach ensures that a running synchronization is guaranteed to finish even if you manually unload the app from the device memory.
2. On the **iOS** platform, the app works in the main `webView`, while the background synchronization uses the second `webView`. This ensures that the user can continue working with the app while the structure synchronization is in progress. Unlike the Android platform, the synchronization can be interrupted when you unload the app manually or if the iOS platform unloads it.

The iOS app uses `WKWebView` instead of `UIWebView`. As such, the app uses Cordova 6.1.1 framework and supports only iOS 11 and later.

`WKWebView` features are as follows:

- Do not use absolute paths for resources, scripts, iframe containers, etc.
- Do not use cross-domain URLs for resources, scripts, iframe containers, etc.
- `localStorage` data is saved when switching to `WKWebView`.
- We do not recommend using an iframe.

Learn more: official vendor documentation.

If you need to insert a link to a local file into the page, convert the path using the `Terrasoft.util.toUrlScheme()` method.

Example that inserts a link to a local file into the page

```
this.element.setStyle('background-image', 'url("' + Terrasoft.util.toUrlScheme(value) +'")');
```

View the link examples below.

Incorrect link

```
file:///var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C271D9871/Documents/BP
```

Converted link

```
app://localhost/_app_file_/var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C27
```

Features of the `inappbrowser` plugin are as follows:

- All paths must be relative and located inside the website root folder.

  ```
  <img src="images/2.jpg">
  ```

- Do not use cross-domain URLs for resources, scripts, iframe containers, etc.

  ```
  <img src="https://www.worldometers.info/img/worldometers-logo.gif">
  <img src="file:///var/mobile/Containers/Data/Application/DE283C57-94BE-4116-980A-020C271D9871
  ```

- Set the absolute path when you open the website.

  ```
  cordova.InAppBrowser.open("file:///var/mobile/Containers/Data/Application/DE..9871/Documents/
  index.html", "_blank");
  ```

Opening the website using the `inappbrowser` plugin has features similar to the `WKWebView` features.

Resolve synchronization conflicts automatically

During synchronization of a mobile app that runs in offline mode, situations when Creatio cannot save the transferred data can occur due to the following reasons:

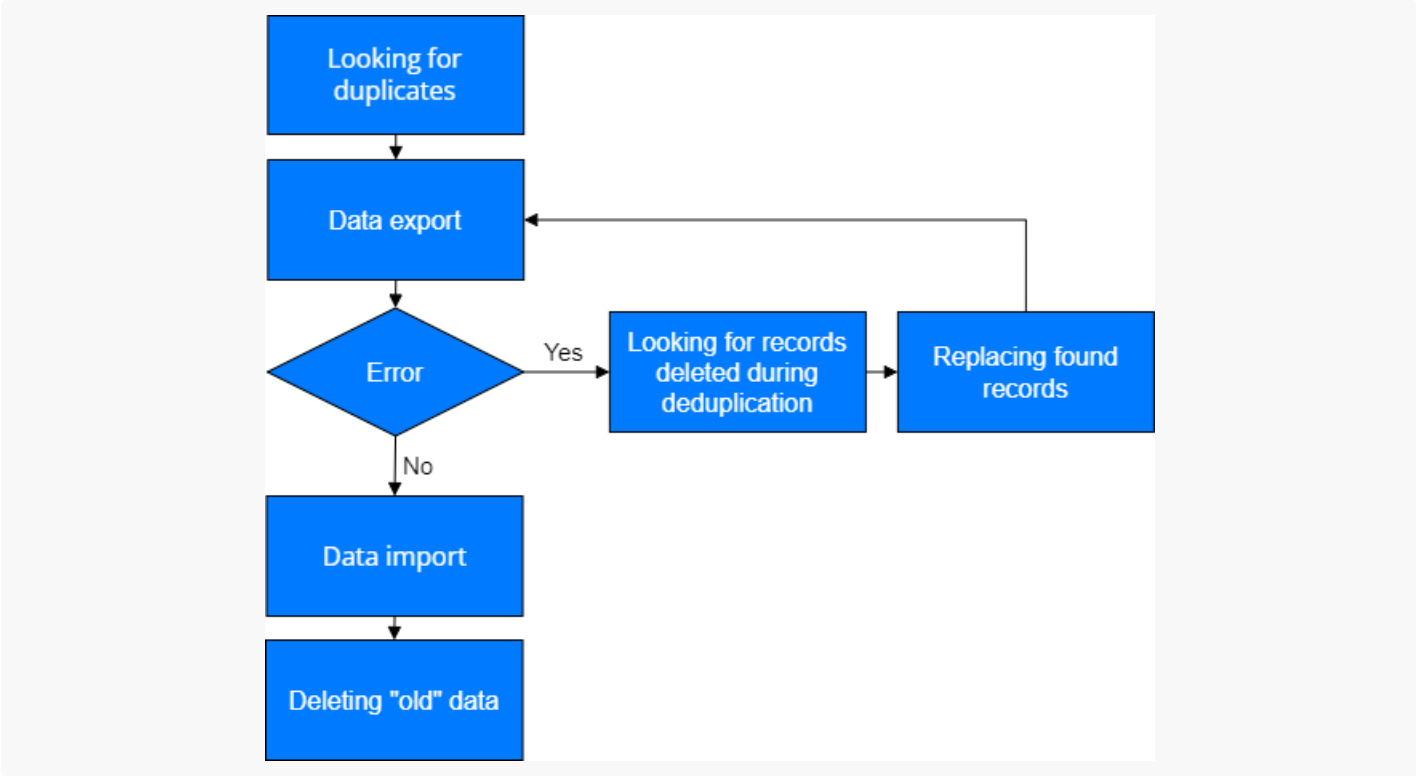
- A record was merged with another duplicate record in Creatio and therefore no longer exists.
- A record was removed from Creatio.

The mobile app handles every conflict automatically.

Merge duplicates

View the procedure to resolve conflicts caused by records deleted from Creatio due to duplicate merging in the

figure below.



During the synchronization, the app first retrieves data about the merged records since the last synchronization from the server. Specifically, which records were deleted, and which records replaced them. If no errors occur during the export, Creatio executes the import. If an error related to the foreign key exception (`Foreign Key Exception`), or an error related to a record not being found on the server (`Item Not Found Exception`) occurs, the following procedure to resolve this conflict is executed:

- Creatio searches the exported data for columns that contain the old record.
- Creatio replaces the old record in the found columns with the new record into which data was merged.

Then the record is re-sent to Creatio. When the import finishes and information about merged duplicates becomes available, the old records are deleted locally.

Record not found

When the server returns an error about the record the user modified not being found in Creatio, the app executes the following actions:

1. Check if the record exists in the list of records deleted as part of the duplicate merge.
2. If the record is not in the list of deleted records, delete the record locally.
3. Remove information about this record from the synchronization log.

Thus, the app considers this conflict resolved and continues data export.

BaseConfigurationPage class JS

Advanced

Page controller classes are inherited from the `Terrasoft.controller.BaseConfigurationPage` class that provides methods of handling the life cycle events.

Methods

`initializeView(view)`

Method is called after the page view in the DOM is being created (but was not rendered). On this stage you can subscribe to the events of the view classes and perform additional actions with DOM.

`pageLoadComplete(isLaunch)`

Provides extension of the logic that is executed at the page load and return. The `true` value of the `isLaunch` parameter indicates that the page is being loaded for the first time.

`launch()`

Called only when the page is opened. The method initiates the loading of data. If you need to load additional data, use the `launch()` method.

`pageUnloadComplete()`

Provides extension of the logic that is executed at the page unload and closure.

PageNavigator class JS

Advanced

The `Terrasoft.PageNavigator` class manages the life cycle of the pages. The class enables opening and closing of the pages, updating of the irrelevant data and storing the page history.

Methods

`forward(openingPageConfig)`

Method opens page according to the properties of the `openingPageConfig` configuration parameter object. Main properties of this object are listed in the table.

The `openingPageConfig` object properties

`controllerName`	name of the controller class
`viewXType`	view type according to xtype
`type`	page type from the `Terrasoft.core.enums.PageType` enumeration
`modelName`	name of the page model
`pageSchemaName`	name of the page schema in configuration
`isStartPage`	flag indicating that the page is a start page. If previously the pages have been opened, they will be closed
`isStartRecord`	flag indicating that the view/edit page should be the first after the list. If there are other opened pages after the list, they will be closed
`recordId`	Id of the record of the page being opened
`detailConfig`	settings of the standard detail

`backward()`

The method is closing the page.

`markPreviousPagesAsDirty(operationConfig)`

Method marks all previous pages as irrelevant. After returning to previous pages, the `refreshDirtyData()` method is called for each page. The method re-loads the data or updates the data basing on the `operationConfig` object.

`refreshPreviousPages(operationConfig, currentPageHistoryItem)`

Method re-loads data for all previous pages and updates the data basing on the `operationConfig` object. If the value is set for the `currentPageHistoryItem` parameter, the method performs the same actions for the previous pages.

`refreshAllPages(operationConfig, excludedPageHistoryItems)`

Method re-loads data for all pages or updates the data basing on the `operationConfig` object. If the `excludedPageHistoryItems` parameter is set, the method does not update the specified pages.

Router class JS

Advanced

Routing is used for managing visual components: pages, pickers, etc. The route has 3 states:

1. `Load` – opens a current route.
2. `Unload` – closes current route on return.
3. `Reload` – restores the previous route on return.

The `Terrasoft.Router` class is used for routing and it's main methods are `add()`, `route()`, `back()`.

Methods

`add(name, config)`

Adds a route.

Parameters

`name`	unique name of the route. In case of re-adding, the latest route will override the previous one
`config`	describes names of the functions that handle route states. Handlers of the route states are set in the `handlers` property

Example of method use

```
Terrasoft.Router.add("record", {
  handlers: {
     load: "loadPage",
     reload: "reloadPage",
     unload: "unloadLastPage"
  }
});
```

`route(name, scope, args, config)`

Starts the route.

Parameters

`name`	name of the route
`scope`	context of the function of the state handlers
`args`	parameters of the functions of the state handlers
`config`	additional route parameters

Example of method use

```
var mainPageController = Terrasoft.util.getMainController();
Terrasoft.Router.route("record", mainPageController, [{pageSchemaName: "MobileActivityGridPag
```

`back()`

Closes current route and restores previous.