MobileNAV
ERP anywhere

MobileNAV – 4.9 Upgrade Guide from 4.8

This document describes the upgrade steps for MobileNAV Add-on to 4.9 version, if your current version is 4.8. If your Add-on version is earlier (4.7 or below), then you need to execute the proper upgrade guide to 4.8 first.

Note: if your system doesn't have any customizations neither in Add-on nor in Configuration, then you can also delete all MobileNAV objects, and install a new 4.9 system.

To upgrade your MobileNAV installation to 4.9 do the following:

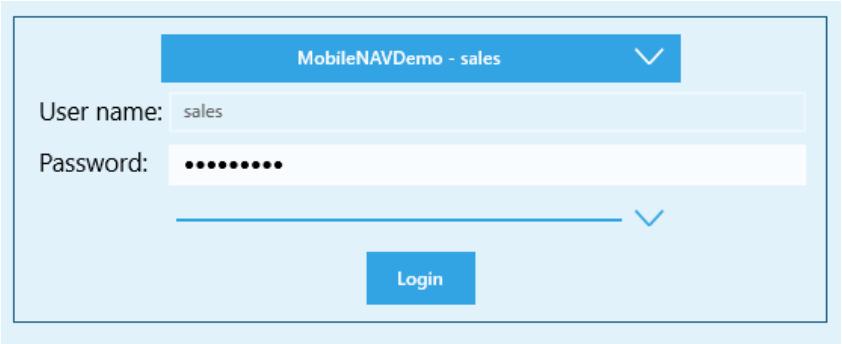
- If you have any Add-on customizations, you have to merge your customizations into new Add-on before you start upgrade
- Point the date of the upgrade
- Inform all of your MobileNAV users, about upgrade
- Stop the NAV services
- Uninstall MobileNAV Components
- Reinstall the new MobileNAV Component Setup
- Import the new Add-on objects
- Compile all MobileNAV objects (Version list filter: *MN*)
- Export and import your configuration (this will recreate the MobileNAV roles)
- Setup NAV roles for MobileNAV users, configurators or offline admins
- Restart the NAV services
- Login into RTC client, and go to General Setup
- Automatic version upgrade will take place
- Log in with MobileNAV client app to test the upgrade

MobileNAV – 4.9 What's new?

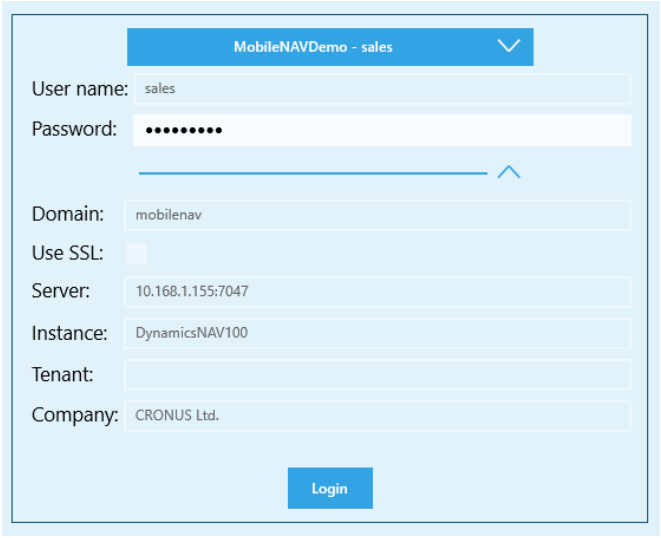
Client app and Add-on improvements

- **Simplified Login screen:** savable Login Configurations were not well-known so far, because the MobileNAV menu of the app was simply not found by users. Also, once the user has provided the login details, he was not interested in the details anymore. Now we have created a "combo box" for Login Configurations, so the user can switch login configurations much easily. Furthermore, now it is possible to edit, create, import, copy and delete Login Configurations. You can generate Login Configuration XML from the MobileNAV User Setup, so you can send the file with login details to the MobileNAV users. We have placed the details of the login on an expandable section, so by default it is collapsed, but the user can still check it, if necessary.

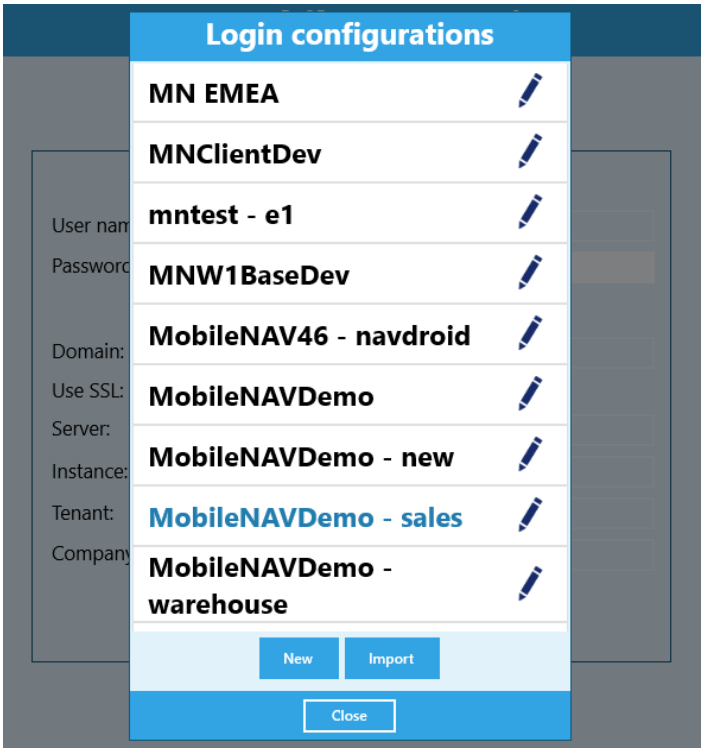
New Login screen with last used Login Configuration selected:



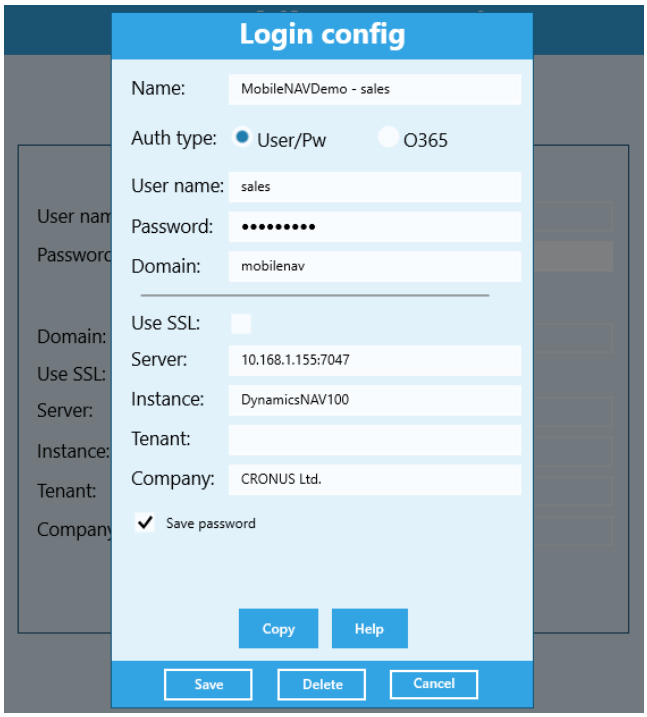
Details expanded:



Login Configurations selector:

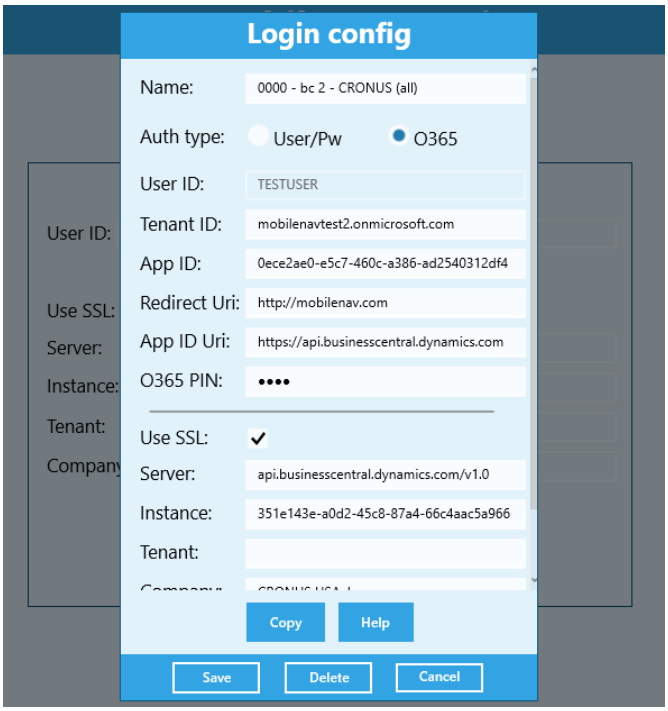


Login Configuration editor:

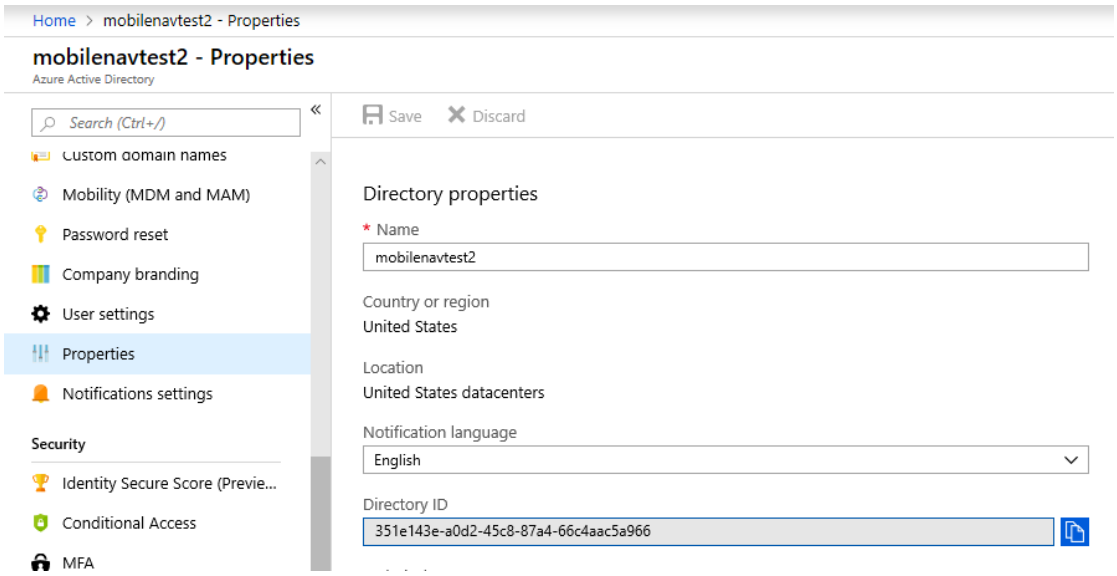


- **Office 365 authentication:** MobileNAV app now supports Offlice 365 authentication as well, not just Windows (NTLM) and NAVUserPassword authentication. For more information how to configure your NAV instance for Office 365 authentication, read this article: https://docs.microsoft.com/en-us/dynamics-nav/authenticating-users-with-azure-active-directory

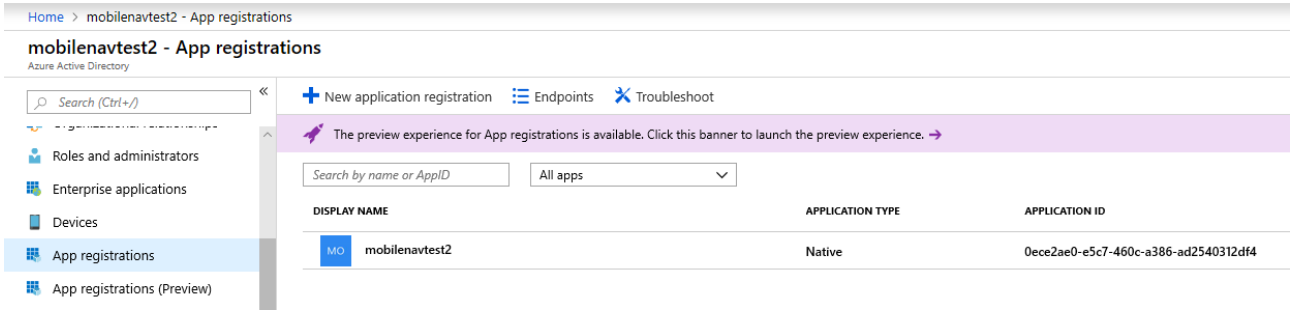
  For Office 365 authentication you need to provide the following:
  - Tenant ID
  - App ID
  - Redirect Uri
  - App ID Uri



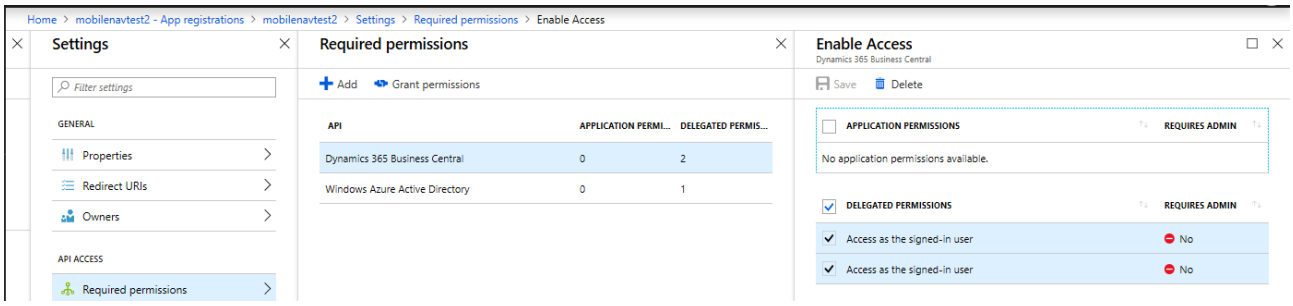
- **<u>Tenant ID:</u>** this is the URL friendly name or the GUID of the Azure Active Directory. You can find this under
  - Azure Active Directory -> Properties -> Name (like: mobilenavtest2.onmicrosoft.com) or
  - Azure Active Directory -> Properties -> Directory ID (like: 351e143e-a0d2-45c8-87a4-66c4aac5a966)



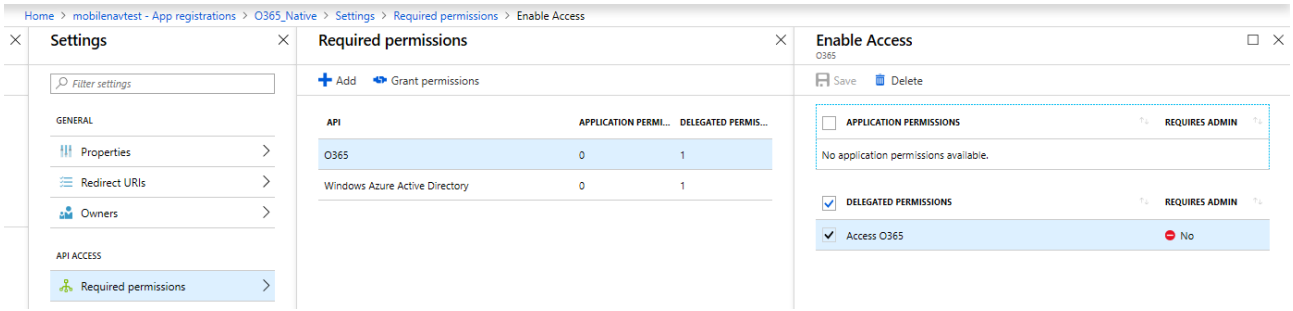
- **App ID:** this is the Application ID (GUID) of the Native App registration of the Azure Active Directory. It is important that the App registration's Application Type is Native, and not "Web app / API". You can find this under Active Directory -> App registrations.



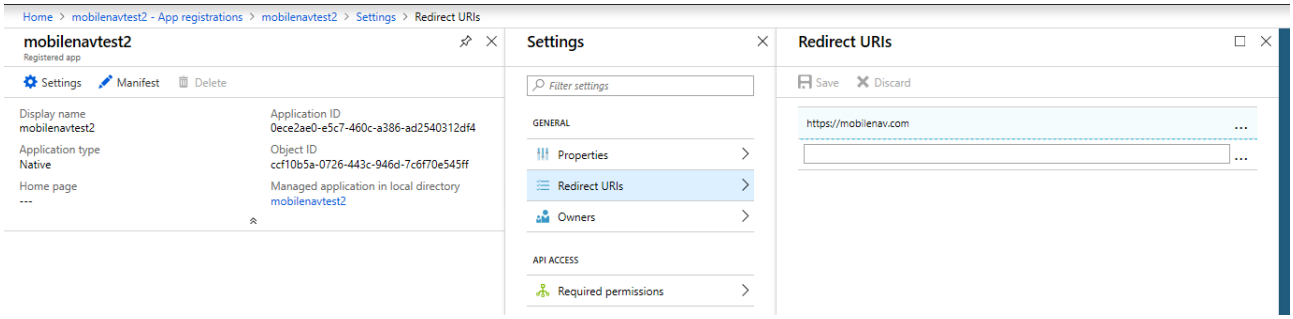
  - In case of Business Central SaaS, this App registration needs to have permission to "Dynamics 365 Business Central". You can find this under Active Directory -> App registrations -> {Native App registration} -> Settings -> Required permissions.



- In any other case, this App registration needs to have permission to a “Web app / API” type App registration. You can find this under Active Directory -> App registrations -> {Native App registration} -> Settings -> Required permissions.



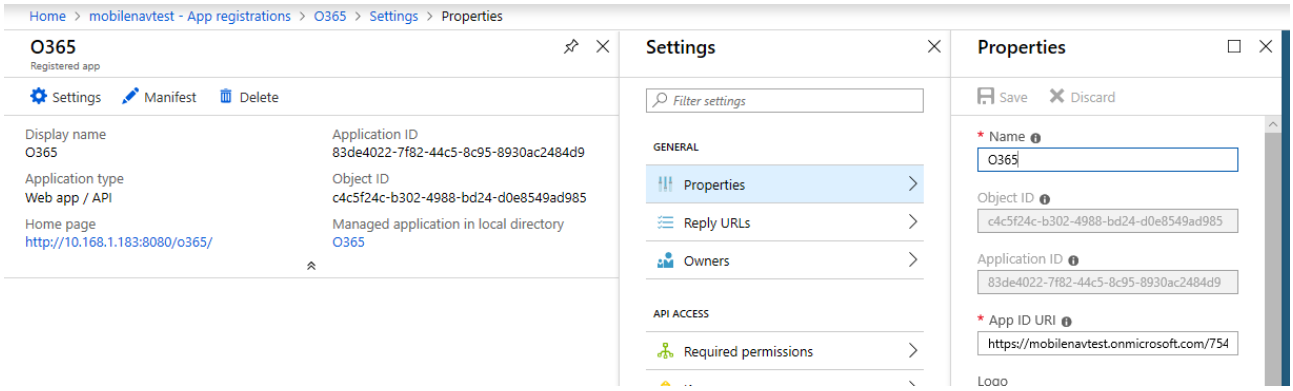
- **Redirect Uri:** this is (one of) the Redirect URI of the Native App registration specified in App ID. You can find this under Active Directory -> App registrations -> {Native App registration} -> Settings -> Redirect URIs. The Redirect URI can be anything, like “https://mobilenav.com”.



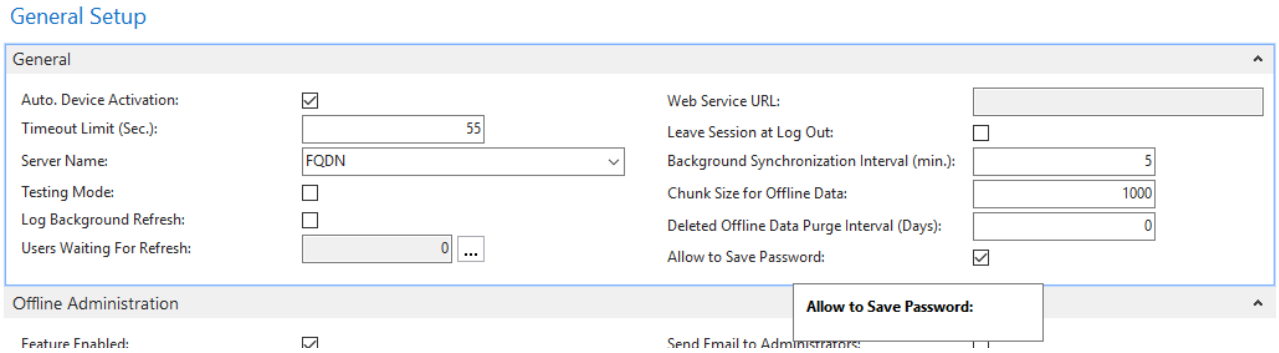
- **App ID Uri:** this is the App ID Uri of the “Web app / API” type App registration.
  - In case of Business Central SaaS, there is a built-in “Web app / API” App

registration, and you need to enter: “https://api.businesscentral.dynamics.com”

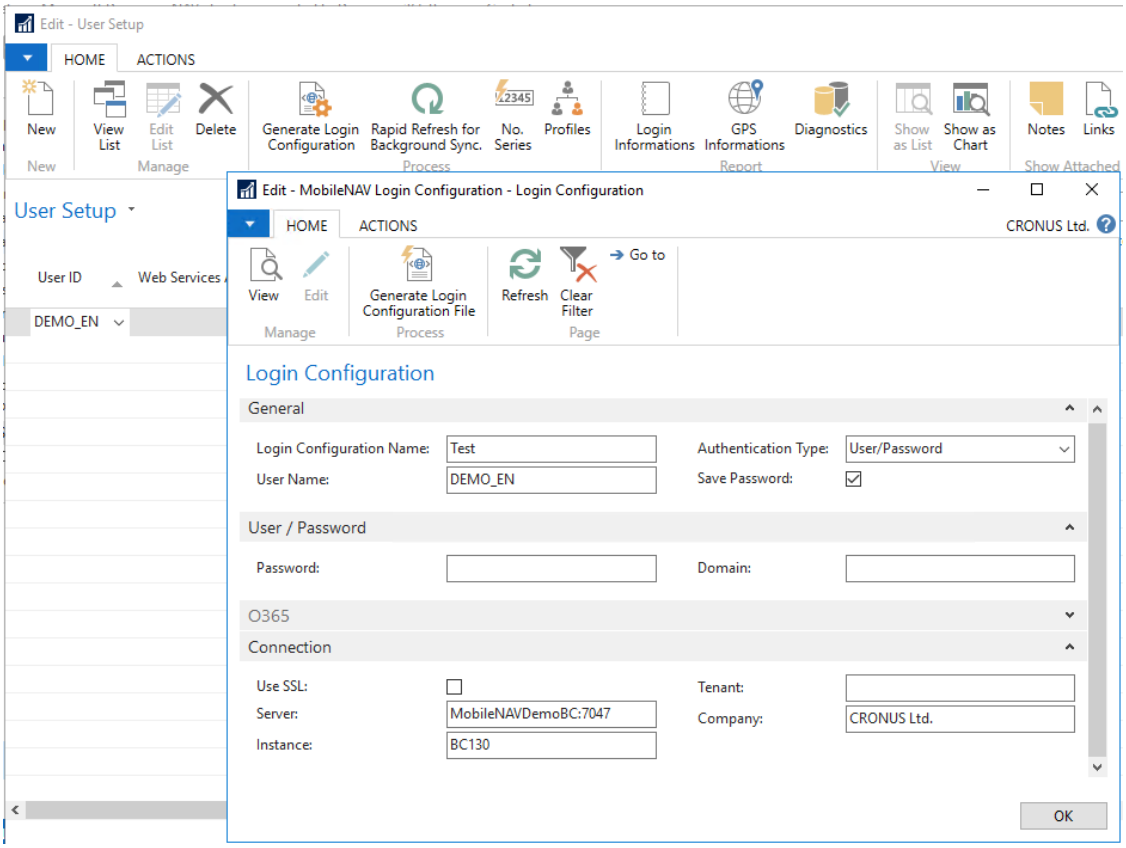
- In case of any other type of NAV, you have a “Web app / API” type App registration, and you can find this information under: Active Directory -> App registrations -> {Web app /API App registration} -> Settings -> Properties -> App ID Uri.



- **<u>Allow to Save password:</u>** now you can control on the server side, whether you allow users to save their passwords. Because of backward compatibility, the default is “on”, but you can turn it “off”, and then the users should provide their passwords every time they log in.



- **<u>Generate + Import Login Configuration:</u>** now you can generate Login Configuration XML file in the MobileNAV User Setup for a user. If you send the file to the device, then you can also import the file in the Login Configurations selector window with the Import button. Depending on the platform, the MobileNAV app can also be automatically launched when opening the “.mnlc” file.



- **Assisted Setup:** we have created a brand-new wizard to speed up the initial setup of MobileNAV. The Assisted Setup guides you thru the mandatory steps after the installation.

- **"By Trigger" option for Type of Check for Changes:** when the MobileNAV client refreshes offline data, then the MobileNAV Addon tells the client app which offline pages has to be downloaded, otherwise the client would download all offline pages from scratch with every refresh. When does this "calculation" happen, can be controlled by the "Type of Check for Changes" option. In earlier version there were 2 supported types:
    - "By Client": means that the recalculation is executed at the beginning of the offline data refresh by the client. Since this enlongers the refresh duration, it is advised to use this only for offline pages where it is necessary to download the refresh data as soon as possible. If you are using too many pages with this type, then "offline timeout" can happen.
    - "By NAS": means that the recalculation is executed by a Job Queue (by NAS) in the background periodically, and during refresh the client just receives the last result of the background recalculation. You can control how often this calculation should run with the "Check for Changes Period (Hours)".

The new “By Trigger” option means that there is no recalculation, and the offline page level timestamp or the record level timestamp is maintained by custom code (by trigger).

From version NAV 2016, we have implemented a codeunit called “MN ByTrigger Subscriptions”, which subscribes to the OnAfterOnDatabaseInsert/Update/Delete/Rename events for tables for which there is an offline page with “By Trigger” option. These events fires when a record has been modified. In the event handler, the “trigger” updates the page level timestamp (and the record timestamp in case of record level synchronization), so next time the MobileNAV client app refreshes offline data, he will see that he needs to download this offline page (or just the changed records). This implementation has certain limitations, which you need to consider before setting an offline page to “By Tigger”:

- FlowFields
- Global variables
- Fields from other table
- User specific filtering
- FlowFilters

In these cases you need to make sure that you update the page (or record) also when these FlowFields, Globals, etc. would change, and its new value should be downloaded to the offline client.

- **Refresh After for Offline pages:** when you have a classical “parent-child” relation, like document header + lines, and you have 2 offline pages for those, then it does not make sense to set up different offline “recalculation strategy” for them. With “Refresh After”, you can signal on the “child” page, that the recalculation of this “child” offline page should happen right after the specified “parent” page. For example you can specify the “Refresh After” as “MNSalesOrder” on the “MNSalesOrderLine” page, and as a result the “recalculation” of “MNSalesOrderLine” will happen right after recalculation of “MNSalesOrder”.

- **Filter By Parent:** when you have a classical “parent-child” relation, like document header + lines, and you have 2 offline pages for those, then sometimes it is hard to filter the “child” page properly only to download those records, where the “parent” record is downloaded. Since there is an offline recalculation functionality in the MobileNAV Addon, the Addon already “knows” which offline “parent” records are downloaded for a certain user. Now we have created helper functions which you can utilize in the OnOpenPage trigger of the “child” page. For example, if you want to filter the MNSalesOrderLine page by the “parent” MNSalesOrder page, then code looks as follows for the OnOpenPage trigger of MNSalesOrderLine page:

```
IF MobileNAVObjectFunctions.OfflineFilterByParentEnabled(CurrPage.OBJECTID(FALSE), PAGE::"MN SalesOrder") THEN
BEGIN
  CLEARMARKS();

  MobileNAVObjectFunctions.GetOfflineRecords(PAGE::"MN SalesOrder", MobileNAVOfflineAdmin);
  IF MobileNAVOfflineAdmin.FINDSET(FALSE, FALSE) THEN
  REPEAT
    SETRANGE("Document No.", MobileNAVOfflineAdmin.Key2);
```

```
    IF FINDSET(FALSE, FALSE) THEN
    REPEAT
      MARK(TRUE);
    UNTIL NEXT = 0;
  UNTIL MobileNAVOfflineAdmin.NEXT = 0;
  SETRANGE("Document No.");

  MARKEDONLY(TRUE);
END;
```

The code works the following way:

1. Call “OfflineFilterByParentEnabled” with the current page object ID + the “parent” page ID to check whether the current page and the “parent” page is offline and the “Filter By Parent” is set to TRUE
2. Clear marking
3. Call “GetOfflineRecords” to retrieve MobileNAV Offline Admin records of the “parent” page and the current user
4. Iterate thru the parent MobileNAV Offline Admin records
5. Filter the current records by parent record’s key(s)
6. Iterate thru the result, and mark them one by one
7. Remove the filtering
8. Call MARKEDONLY(TRUE) to show only the marked records on the page

The important part is the filtering by parent record keys. MobileNAV Offline Admin will store the first 3 key field of the primary key of the parent record, and you can use those to filter the “children”. For example, in this case for MNSalesOrder type records the MobileNAVOfflineAdmin.Key1 is the “Document Type”, MobileNAVOfflineAdmin.Key2 is the “No.”, so you can use the Key2 to filter the lines by “Document No.”

- **Report parameters:** from NAV 2015 or above, you can configure reports more easily. In case of “Page Type” = “Report”, you can specify the “Report ID”, which binds the MobileNAV request page with the report object. Furthermore, you bind the request page fields to report parameters or DataItem filter fields. MobileNAV Addon provides a new codeunit called “MobileNAV Report Helper”, which provides methods to easily construct a special XML which standard Reports can process as parameters, so you don’t need to modify the Report object to process the report generation parameters.

MN CustomerTop10 Report

General

Object ID:	42012966
Page Type:	Report
Main Menu Action:	Create
Service Name:	MNCustomerTop10Report
Name:	MN CustomerTop10 Report
Localized Name:	Customer - Top 10 List
Report ID:	111
Table No.:	18

Field Name	Localized Field Name	Report Field Source	Report Field Name
Number	Number		
custNo	Customer No.	Customer	No.
noOfRec	Quantity	<Report Parameter>	NoOfRecordsToPrint
showType	Show	<Report Parameter>	ShowType
chartType	Chart Type	<Report Parameter>	ChartType
ShowAsPDF	Show as PDF		

```
22 MNCustomerTop10Report(CustNo : Text[250];NoOfRec : Integer;ShowType : 'Sales (LCY),Balanc
23 CLEAR(Language);
24 GLOBALLANGUAGE(Language.GetLanguageID(MobileNAVUserSetup.GetUserLanguageCode(USERID)));
25
26 MobileNAVReportHandler.Initialize('MNCustomerTop10Report');
27 MobileNAVReportHandler.AddByPageField('custNo', FORMAT(CustNo, 0, 9));
28 MobileNAVReportHandler.AddByPageField('noOfRec', FORMAT(NoOfRec, 0, 9));
29 MobileNAVReportHandler.AddByPageField('showType', FORMAT(ShowType, 0, 9));
30 MobileNAVReportHandler.AddByPageField('chartType', FORMAT(ChartType, 0, 9));
31 MobileNAVReportHandler.Generate(SaveAsFormat::PDF, Base64Result);
```

Base configuration improvements

- **“Refresh After” and “Filter By Parent Enabled”:** we have extended the “child” pages with a code to filter by “parent” page. It is important to note, that the code checks whether both “child” and “parent” pages are offline pages, and the “Filter by Parent Enabled” is set to true for the “child” page. Since it uses marking mechanism for the records, and this technique does only works well, if you filter the “parent” page in a way, what you cannot copy that filter easily to “child” page, and via enabling this filtering, you can limit the number of “child” records drastically.

  This technique also requires that the “recalculation” of “parent” page is earlier than the “child” page, so for these “child” pages we have also set the “Refresh After” to the “parent” page name.

These are the list of pages, which are prepared with the code for filter by parent (the ones, where Filter By Parent Enabled is set to TRUE are **bold**):

- **MNInventoryPickLine** (by MNInventoryPick)
- **MNInventoryPutawayLine** (by MNInventoryPutaway)
- MNItemCrossReference (by MNItem)
- MNItemUnitOfMeasure (by MNItem)
- MNItemVariant (by MNItem)
- MNSalesLineDiscount (by MNItem)
- MNSalesPrice (by MNItem)
- MNJobPlanningLine (by MNJobTask)
- MNJobTask (by MNJob)
- MNPurchaseOrderLine (by MNPurchaseOrder)
- MNProdOrderRoutingLine (by MNReleasedProdOrderLine)
- MNReleasedProdOrderLine (by MNReleasedProdOrderLine)
- MNResourceUnitOfMeasure (by MNResource)
- MNSalesInvoiceLine (by MNSalesInvoice)
- MNSalesOrderLine (by MNSalesOrder)
- MNSalesQuoteLine (by MNSalesQuote)
- MNSalesReturnOrderLine (by MNSalesReturnOrder)
- MNServiceItemComp (by MNServiceItem)
- MNServiceItemLine (by MNServiceHeader)
- MNServiceHeader (by MNMyServiceTask)
- MNServiceLine (by MNServiceHeader)
- MNServiceResourceAlloc (by MNServiceHeader)
- MNShipToAddress (by MNCustomer)
- MNTimeSheetLine (by MNTimeSheetHeader)
- MNTransferOrderLine (by MNTransferOrder)
- MNTroubleshootingLine (by MNTroubleshootingSetup)
- **MNWarehouseMovementLine** (by MNWarehouseMovement)
- **MNWarehousePickLine** (by MNWarehousePick)
- **MNWarehousePutawayLine** (by MNWarehousePutaway)
- **MNWhseReceiptLine** (by MNWhseReceipt)
- **MNWhseShipmentLine** (by MNWhseShipment)