

A large, light blue decorative graphic consisting of a circle at the top, a thin line extending downwards, and a wide, curved swoosh at the bottom, resembling a stylized 'i' or a navigation arrow.

Navigate Pro – Partner Extension V100.0.4.0

Info Studi S.r.l.

Via L. Ariosto 21 – 20091 Bresso (MI)
Tel. +39 02 6103411 – Fax +39 02 61034190
www.infostudi.it - infostudi@infostudi.it

Sede di Vicenza

Via Istria 6/8 – 36023 Longare (VI)
Tel. +39 0444 1805350 – Fax +39 02 61034190

P.IVA 00924450968 - C.F. 08764250158

REA MI 1248744 - Registro Imprese MI149-36561 Capitale sociale Euro 52.000,00 i.v.

Sommario

Abstract	3
Adding Navigate Pro as a dependency	4
Subscribing Navigate Pro Events	5
Appending documents Step by Step.....	5
Json data structures.....	7
Document part.....	7
Relationship Part.....	8
Customizing Right-Click	10
Adding information to Tabular Output	11

Abstract

In this document you can find a new method, to submit your own data to Navigate Pro graphical engine. You can now choose to replace the whole dataset or append your own documents to an existing searching result. It is available from Navigate Pro version 100.0.4.0.

The main goal of this document is to show partner:

- How to extend Navigate Pro app;
- How to subscribe to events, skipping the original ones;
- How to send a new set of data to Navigate Pro engine;
- How to modify the original set of data of Navigate Pro.

All the examples are referred to a demo a codeunit, "codeunit 50137 IST_NPPartnerNavigateProMgtV2", that shows a way to interface the original app.

All the images are related to a Visual Studio Code (VSC) project having runtime equal to 7.

```
"runtime": "7.0"
```

To use this feature on Cloud you need to subscribe at least "Advanced" plan.

Adding Navigate Pro as a dependency

To extend Navigate Pro, using this document, you need to add the original app in your *app.json* file.

```
"dependencies": [{  
  "id": "acff63be-8808-4efe-a0e8-e062c4a71b9a",  
  "publisher": "Info Studi",  
  "name": "Navigate Pro",  
  "version": "100.0.4.0"  
}],
```

Subscribing Navigate Pro Events

Starting from version **100.0.4.0** Navigate Pro (NP) gives you 3 ways to launch your functionality:

1. Subscribing original NP buttons on documents;
2. Adding your own NP buttons.
3. Catching the result of searching routines and replacing/modifying/appending data to the original flow.

In the first 2 cases you will have to rewrite the whole searching routine, because it is not predictable how new documents can interact with the original data.

Here we introduce a way to realize point 3, that relies on getting original data through a json file and modifying it.

Appending documents Step by Step

1. To add your own document type:
 - a. Extended enum 70104171 **IST_NPDocumentType**, adding your own document. Be careful to start your ID from 50000, to avoid problems with further releases.
 - b. Subscribe to event *"OnAfterOnOpenDocument"* on table **IST_NPTabularOutput** to choose which documents will be open from *"TabularOutputPage"*.

```

//This event is used to open your customized document in tabular Output
[EventSubscriber(ObjectType::Table ,Database::IST_NPTabularOutput,'OnAfterOnOpenDocument','',false,false)]
0 references
local procedure OnAfterOnOpenDocument(Rec: record IST_NPTabularOutput)
var
    SalesHeader :record "Sales Header";
begin
    case Rec."Document Type" of
        Rec."Document Type"::"Partner Sales Quote":
            begin
                //Find your own added document. i.e. Sales Header
                SalesHeader.GetBySystemId(Rec."System ID");
                //Run your own document page
                PAGE.RUNMODAL(PAGE::"Sales order", SalesHeader);
            end;
    end;
end;
end;

```

2. Subscribe to event *"OnBeforeShowingGraph"*, to get the a json containing the standard searching result, in json format (to see how the json is composed watch next chapter)

```

[EventSubscriber(ObjectType::Codeunit,Codeunit::"IST_NP Interface", 'onBeforeShowingGraph', '', false, false)]
0 references
procedure AppendMyOwnDocuments(var InJson: JsonObject; LastEntryNo: Integer; StartingDocRef: RecordRef)

```

With this event you will find 2 more variables:

- LastEntryNo, indicating the overall index of the json (how many documents and relationships are there). Your next document or relationship will start with LastEntryNo+1;
 - StartingDocRef, where you can find all the information from where the app has been triggered.
3. Inside the procedure you should:
 - a. Read the json to extract information:

- i. Find out the documents in the flow.
 - ii. Find out how they are linked (relationship).
 - b. Create your own routine to add your new document, linked to the documents already in the flow, and the relations between your new documents and the existing documents.
4. Subscribe to event “*OnDoubleClickEventJson*” on page **IST_NavigatePro** to open your new added documents. In this procedure you should retrieve the list of the documents, their document type and their SystemID, deciding the right page for each document Page (see image below as for example)

```
[EventSubscriber(ObjectType::Page, Page::IST_NavigatePro,'OnDoubleClickEventJson','', false, false)]
0 references
procedure PartnerDoubleClick(iEntryno:Integer; jsonData:JsonObject)
//Use this procedure to open selected documents
//bIsHandle is used to avoid running both types of personalization. bIsHandle must be set true by old version c
//response routines
var
  enumDocumentType: enum IST_NPDocumentType;
  SalesHeader: record "Sales Header";
  guidSystemID: guid;
begin
  //Retrieve Document list, document type, entryno and systemid
  //FillDocumentList(jsonData)

  //Get the selected document
  //GetSelectDocument(var enumDocumentType, var guidSystemID)

  //open document. Note: opening an NP "document Type", you will fire 2 pages, the original and your custome
  Case enumDocumentType of
    enumDocumentType::"Partner Sales Quote":
      begin
        //Find your own added document. i.e. Sales Header
        SalesHeader.GetBySystemId(guidSystemID);
        //Run your own document page
        PAGE.RUN(PAGE::"Sales Quote", SalesHeader);
      end;
  End;
end;
```

Note: Documents managed by the standard flow of Navigate Pro (Microsoft standard pages) are handle by the application. Adding your own page to standard documents will open 2 (or more) pages.

Json data structures

To send data to the addin engine, you must provide a json file to the application (from here jsonData).

Inside this file there will be a part, with the information you want to draw on the images, and a part to built the lines between the images.

Note: Starting from version 100.0.4.0 the content of json file will be verified by the application to ensure each key-value pair is typed (see note below).

The most important part of the json object is the index (1, 2 etc.), that link together all data. The index, often called entry no., must be without missing numbers.

```

▼ 1:
  sType:           "Document"
  sTableName:      "Sales Order"
  sDocumentNo:    "1003"
  iLevel:         "1"
  sDate:          "26/01/23"
  sDescription:   ""
  iDocType:       1
  sSystemID:      "{716FF242-409C-EB11-AC6B-000D3A2B6372}"
  bDocExists:     true

▼ 2:
  sType:           "Document"
  sTableName:      "Sales Quote"
  sDocumentNo:    "1001"
  iLevel:         "0"
  sDate:          ""
  sDescription:   ""
  iDocType:       0
  sSystemID:      "{00000000-0000-0000-0000-000000000000}"
  bDocExists:     false
  
```

Example of json Data for the images (index is 1,2,3 etc.).

Each image is identified by its index. There are 2 blocks of data inside the json:

- “document”: carries information about what you draw over images. In the original NP, in this part you can see information about documents, but you can draw whatever you want;
- “relationship”: gives the information about how drawing lines.

Document part

In this part of the jsonData you must indicate what the user will see or other information useful to other processes.

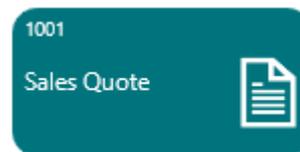


Std Document Image of NP

You must fill all key-value pairs. All the pairs are mandatory. The key is fixed:

- sType: the value is fixed. It must contain "Document". It is not printed on video;
- sTableName: it contains the string that will be printed in the middle position. In NP standard it shows the page name, but you can send any string you want;
- sDocumentNo: it contains the string that will be printed in the first line. In NP standard it shows the "document no.", but you can send any string you want. Up to 20 characters;
- iLevel: it contains an integer (passed as string), from 0 to 5. It indicates the starting position of the image, from left to right. 0 is the far-left column, while 5 is the last column on the right. It must be always filled with an integer like value. The Type is checked by application;
- sDate: it contains the string that will be printed in the last line. It must contain a date (i.e. FORMAT("Document Date")). The Type is checked by application.
- sDescription: Currently not used. For future internal developments;
- bDocExists: to indicate an existing document. Useful to skip opening documents no more existing (i.e. an invoiced Sales Order). The Type is checked by application;
- iDocType: it contains the document type of the application. You can fill it with `format(enum::IST_NPDocumentType::"Sales Order".AsInteger())` instruction (see code example). The Type is checked by application;
- sSystemID: must be filled with the guid of the field SystemID of every record. The Type is checked by application.

Following the json of the previous paragraph the image with index 2 will be:



An image filled with uncomplete data (sales quote has been deleted).

Relationship Part

In this part of the jsonData you must indicate how to connect the image you drew.

```

▼ 5:
  sType:           "Relationship"
  iFatherEntryNo: "1"
  iDocEntryNo:    "2"
  iLinkType:      "0"

▼ 6:
  sType:           "Relationship"
  iFatherEntryNo: "1"
  iDocEntryNo:    "3"
  iLinkType:      "1"

▼ 7:
  sType:           "Relationship"
  iFatherEntryNo: "3"
  iDocEntryNo:    "4"
  iLinkType:      "1"

```

There are 4 Key-value pairs to fill. All the pairs are mandatory. The Key is fixed:

- sType: the value is fixed. It must contain "Relationship". It is not printed on video;
- iFatherEntryNo: it contains an integer (as string). Each line is drawn between 2 documents. The number here is the index of the jsonData from which you start drawing the line (i.e. 1 in this field indicate the document having NEW1 as sTablename). Index must be referred to a Document object. The Type is checked by application;
- iDocEntryNo: it contains an integer (as string). The number here must be referred to a Document and indicate the arrow part of the relationship. The Type is checked by application;
- iLinkType: it contains an integer (as string). It must be 0, full line, or 1, dashed line. Putting here different numbers can lead to problems in drawing lines. The Type is checked by application.



The line between document with index 1 and the document with index 2 in the previous json image. The iLinkType is 0, producing a full line.

Note: The entries numbers, written in the relationship part, must refer to the json index of documents. If the index number does not exist, or it is wrong (i.e. it refers to a relationship), the drawing will not be drawn or will be drawn with missing documents.

Customizing Right-Click

With App version 100.0.4.0 the right click functionality is changed, increasing the customization possibilities. Now the new event exposes both the Document Type of the application and the SystemID of the record.

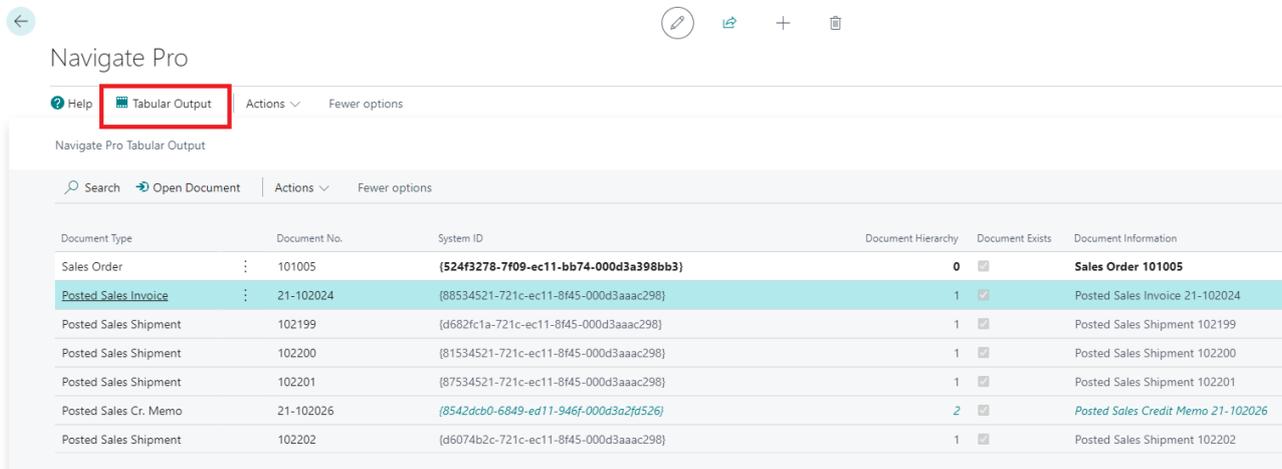
```
[EventSubscriber(ObjectType::Page, Page::IST_NavigatePro, 'OnRightClickEventDocTypeSystemID', '', false, false)]
0 references
local procedure RightClickEventDocTypeSystemID(enumDocumentType:enum IST_NPDocumentType; guidSystemID:guid; jsonData:JsonObject);
var
begin
    Message('Right Click New way Document Type %1 System Id %2', enumDocumentType, guidSystemID);
end;
```

Subscribing right-click event.

A further useful parameter of the event is the json object shared with the drawing area, permitting to receive all the information drawn.

Adding information to Tabular Output

Version 100.0.4.0 of NP introduces a new possibility to visualize the searched data.



Document Type	Document No.	System ID	Document Hierarchy	Document Exists	Document Information
Sales Order	101005	{524f3278-7f09-ec11-bb74-000d3a98bb3}	0	<input checked="" type="checkbox"/>	Sales Order 101005
Posted Sales Invoice	21-102024	{88534521-721c-ec11-8f45-000d3aaac298}	1	<input checked="" type="checkbox"/>	Posted Sales Invoice 21-102024
Posted Sales Shipment	102199	{d682fc1a-721c-ec11-8f45-000d3aaac298}	1	<input checked="" type="checkbox"/>	Posted Sales Shipment 102199
Posted Sales Shipment	102200	{81534521-721c-ec11-8f45-000d3aaac298}	1	<input checked="" type="checkbox"/>	Posted Sales Shipment 102200
Posted Sales Shipment	102201	{87534521-721c-ec11-8f45-000d3aaac298}	1	<input checked="" type="checkbox"/>	Posted Sales Shipment 102201
Posted Sales Cr. Memo	21-102026	{8542dcb0-6849-ed11-946f-000d3a2fd526}	2	<input checked="" type="checkbox"/>	Posted Sales Credit Memo 21-102026
Posted Sales Shipment	102202	{d6074b2c-721c-ec11-8f45-000d3aaac298}	1	<input checked="" type="checkbox"/>	Posted Sales Shipment 102202

Pressing “Tabular Output” NP shows all the documents in a tabular way, exportable in Excel.

The Table (IST_NPTabularOutput) and the Page (IST_NPTabularOutput) are both extensible, so a partner can add his own fields, to better fulfill customer requests.

To fill the added fields, you must subscribe to event “*onBeforeOpeningPageTabularOutput*”.

```
//This event is used to add content to customized fields on Tabular output table
[EventSubscriber(ObjectType::Codeunit ,Codeunit::"IST_NP Interface", 'onBeforeOpeningPageTabularOutput', '', false, false)]
0 references
local procedure OnAfterOnOpenDocument2(var tTmpTabularOutput: record IST_NPTabularOutput)
var
    SalesHeader :record "Sales Header"; //Just as an example
begin
    IF not tTmpTabularOutput.findset then exit;

    repeat
        case tTmpTabularOutput."Document Type" of
            tTmpTabularOutput."Document Type": "Partner Sales Quote":
                begin
                    tTmpTabularOutput."External Document No." := 'Customer Content 123';
                    tTmpTabularOutput."Special Requests"      := 'Ship all together';
                    tTmpTabularOutput."Sent to Cloud Archive" := False;
                end;

            tTmpTabularOutput."Document Type": "Sales Order Archive":
                begin
                    tTmpTabularOutput."External Document No." := 'Vendor Content 123';
                    tTmpTabularOutput."Special Requests"      := 'Use one pallet';
                    tTmpTabularOutput."Sent to Cloud Archive" := True;
                end;
        end;
        tTmpTabularOutput.modify;
    until tTmpTabularOutput.next=0;
end;
```

Simple way to fill added fields for the Tabular Output. Normally, for each document type, there would be a specific routine.