



Microsoft Dynamics NAV 2016 Developer

Courseware

Content

What's new in the C/AL Editor	4
Proper syntax highlighting	5
Line numbers	5
Change indicators	5
Auto-complete intellisense style	6
Syntax Tooltips	6
Table Definition from the Code Editor	6
Undo	7
Block changes	7
The "useoldeditor" Option	8
Events	10
What Are Events?	11
How Events Work	11
Event types	12
Event areas	13
How to Implement Events	19
Publishing an event	19
Raising an event.....	21
Subscribing to Events.....	22
Example of an Event Subscriber Function	22
Extensions.....	26
What are extensions?.....	27
How extensions work	27
Lifecycle of an Extension	28
Windows PowerShell Cmdlets.....	30
Extension Packages.....	31
Extension Manifests	31
Develop an Extension	34
Building the extension package.....	36

To sign an extension package	37
Publishing and Installing an Extension	38
Supported functionality in Extension Packages	40
Extension example.....	41
Development	43
The testing.....	44
Creating the navx package.....	48
Publishing and installing the package	50
Testing the App.....	51
Building workflows	53
What are workflows?	54
Create a workflow event	74
Build a workflow response	78
Enable the workflow event in the NAV system	81
Test automation.....	86
How to start using the application test suite	87
Managing test suites	89
Test selection by churn.....	91
Failure analysis	93
Designing pages for the Universal App	95
Designing for Different Screen Sizes on Tablet and Phone	96
Differences and Limitations When Developing Pages for the Microsoft Dynamics NAV Universal App	98
How to: Implement the Camera in C/AL	103
How to: Implement Location in C/AL	106

What's new in the C/AL Editor

- Proper syntax highlighting
- Line Numbers
- Change Indicators
- Intellisense
- Table Definition from the Code Editor
- Undo
- Block changes
- The “useolddeditor” option



Proper syntax highlighting

All keywords are highlighted with BLUE

Hardcoded text is highlighted with RED

Inactive code is GREEN regardless if the code has been disabled with // or the { }.

Local functions are now prefixed with LOCAL.

Events in the code are marked with :: eg. The Host::AddInReady()

Line numbers

Line numbers are implemented in the code to give an indication of where in the code the cursor is situated.

```
40 LOCAL SendCustomer()  
41 IF Cust.FINDFIRST() THEN BEGIN  
42   CustObj := CustObj.Customer();  
43   CustObj.No := Cust."No.";  
44   CustObj.Name := Cust.Name;  
45   {  
46     CustObj.CreditLimit := Cust."Credit Limit (LCY)";  
47     CustObj.LastDateModified := CREATEDATETIME(Cust."Last Date Modified",0T);  
48     CustObj.PricesIncludingVAT := Cust."Prices Including VAT";  
49     CustObj.Email := Cust."E-Mail";  
50   }  
51   CurrPage.Host.SendCustomer(CustObj);  
52 END;  
53  
54 LOCAL FullScreen()  
55 CurrPage.Host.ExecuteScript('demos.fullScreen()');  
56 CurrPage.Host.ApplyStyle('body','background-color: lightskyblue');  
57  
58 Host::AddInReady()  
59 //ApplyStyle();  
60 SendCustomer();  
61 FullScreen();
```

Change indicators

Whenever a change is made the change indicator will mark each line with a yellow line.

The change indicator will change color to green when it has been compiled or saved.

Auto-complete intellisense style

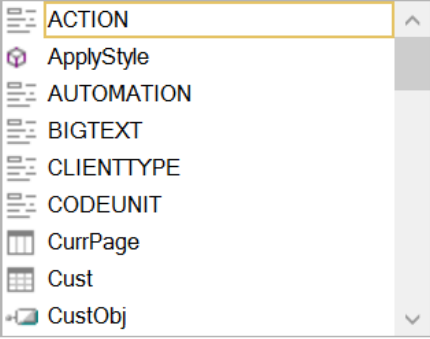
Typing code will open intellisense windows suggesting the relevant field names or actions available.

```

CustObj.No := Cust
CustObj.Name := Cu
CustObj.CreditLimi
CustObj.LastDateMo
CustObj.PricesIncl
CustObj.Email := C
CurrPage.Host.Send

CustObj.GetType().

```



Syntax Tooltips

When selecting a function from the intellisense the syntax will be suggested.

```

InitProgressWindow(SalesHeader);
SalesHeader.CALCFIELDS(|
GetGLSetup;
GetCurrency;

```

[Ok :=] CALCFIELDS(**Field1** [, Field2] ,...)

Table Definition from the Code Editor

Hovering the cursor over a record variable in the editor will open a window showing all fields in the record.

```
SalesSetup."Return Receipt on Credit Memo")
```

Sales & Receivables Setup : 311

Name	Type	Length
Primary Key	Code	10
Discount Posting	Option	4
Credit Warnings	Option	4
Stockout Warning	Boolean	4
Shipment on Invoice	Boolean	4
Invoice Rounding	Boolean	4

```
,TRUE);
```

Undo

As something new the editor will record all changes made while working in the editor enabling undo all the way back to when the object was opened. This also include saving and compiling the object.

Closing the editor will reset the undo records.

This means that to utilize this best the object must be kept open all the time wirking on a change.

Block changes

Sometimes it is necessary to change the variable of many lines at once.

```
4 Customer."No." := OldCustomer."No.";
5 Customer.Name := OldCustomer.Name;
6 Customer."Search Name" := OldCustomer."Search Name";
7 Customer."Name 2" := OldCustomer."Name 2";
8 Customer.Address := OldCustomer.Address;
9 Customer."Address 2" := OldCustomer."Address 2";
10 Customer.City := OldCustomer.City;
11 Customer.Contact := OldCustomer.Contact;
12 Customer."Phone No." := OldCustomer."Phone No.";
```

An example could be that the Customer variable must be changed into the Vendor variable.

Now this is possible by using block change:

Cllice on the first letter of the Customer variable then hold the Alt key down and drag to the last character of the Customer variable on the bottom line.

```
4 Customer."No." := OldCustomer."No.";
5 Customer.Name := OldCustomer.Name;
6 Customer."Search Name" := OldCustomer."Search Name";
7 Customer."Name 2" := OldCustomer."Name 2";
8 Customer.Address := OldCustomer.Address;
9 Customer."Address 2" := OldCustomer."Address 2";
10 Customer.City := OldCustomer.City;
11 Customer.Contact := OldCustomer.Contact;
12 Customer."Phone No." := OldCustomer."Phone No.";
```

Pressing Delete here vill delete the block.

Starting to type Vendor will change all line in one go.

```
3 OnRun()  
4 Ven."No." := OldCustomer."No.";  
5 Ven.Name := OldCustomer.Name;  
6 Ven."Search Name" := OldCustomer."Search Name";  
7 Ven."Name 2" := OldCustomer."Name 2";  
8 Ven.Address := OldCustomer.Address;  
9 Ven."Address 2" := OldCustomer."Address 2";  
10 Ven.City := OldCustomer.City;  
11 Ven.Contact := OldCustomer.Contact;  
12 Ven."Phone No." := OldCustomer."Phone No.";
```

The repeating it on the OldCustomer Variable.

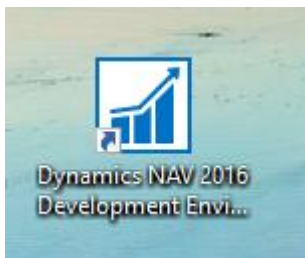
```
4 Vendor."No." := OldVendor."No.";  
5 Vendor.Name := OldVendor.Name;  
6 Vendor."Search Name" := OldVendor."Search Name";  
7 Vendor."Name 2" := OldVendor."Name 2";  
8 Vendor.Address := OldVendor.Address;  
9 Vendor."Address 2" := OldVendor."Address 2";  
10 Vendor.City := OldVendor.City;  
11 Vendor.Contact := OldVendor.Contact;  
12 Vendor."Phone No." := OldVendor."Phone No.";
```

The "useoldeditor" Option

If for some reason the new editor does not agree with you, it is possible to revert to the old editor by starting the development environment up with a parameter:

useoldeditor

Create a new shortcut:



Open properties and add the parameter to the destination field:



Dynamics NAV 2016 Development Environment

Type: Program
Placing: Role Tailored Client
Destination: `\\.\90\Role Tailored Client\finsql.exe" useoldeditor`

And now it is back to normal.

```
Codeunit 80 Sales-Post - C/AL Editor
Documentation()
OnRun(UAR Rec : Record "Sales Header")
OnBeforePostSalesDoc(Rec);

IF PostingDateExists AND (ReplacePostingDate OR ("Posting Date" = 0D)) THEN BEGIN
    "Posting Date" := PostingDate;
    VALIDATE("Currency Code");
END;

IF PostingDateExists AND (ReplaceDocumentDate OR ("Document Date" = 0D)) THEN
    VALIDATE("Document Date",PostingDate);

IF PreviewMode THEN BEGIN
    CLEARALL;
    PreviewMode := TRUE;
    GenJnlPostPreview.Start;
END ELSE
    CLEARALL;

SalesHeader := Rec;
TempServiceItem2.DELETEALL;
TempServiceItemComp2.DELETEALL;
WITH SalesHeader DO BEGIN
    CheckMandatoryHeaderFields(Rec);
    OIOXMLCheckSalesHeader.RUN(Rec);
    IF GenJnlCheckLine.DateNotAllowed("Posting Date") THEN
        FIELDERROR("Posting Date",Text045);

    SetShipInvoiceReceiveFlags(SalesHeader);

    IF NOT (Ship OR Invoice OR Receive) THEN
        ERROR(
            Text020,
            FIELDCAPTION(Ship),FIELDCAPTION(Invoice),FIELDCAPTION(Receive));

    WhseReference := "Posting from Whse. Ref.";
    "Posting from Whse. Ref." := 0;

    IF Invoice THEN
        CreatePrepaymentLines(SalesHeader,TempPrepaymentSalesLine,TRUE);
```

Events

- What Are Events?
- How Events Work
- Event types
- Event areas
- How to Implement Events
 - Publishing an event
 - Raising an event
 - Subscribing to Events
 - Example of an Event Subscriber Function

This chapter is a compilation of available Microsoft material. More information can be found at [https://msdn.microsoft.com/en-us/library/mt299398\(v=nav.90\).aspx](https://msdn.microsoft.com/en-us/library/mt299398(v=nav.90).aspx)





What Are Events?

Dynamics NAV 2016 introduces a new type of integration point – events. Events are a C/AL feature in which an object maintains a list of its subscribers, and notifies them automatically of any state changes. This is done by calling one of the subscriber methods. Subscriber methods need not be in the same object and this fact enables integration developers to build decoupled systems. Dynamics NAV solutions can be written interfering or rewriting much less any of the base application code.

You can use events to design the application to react to specific actions or behavior that occur. Events enable you to separate customized functionality from the application business logic. By using events in the application where customizations are typically made, you can lower the cost of code modifications and upgrades to the original application.

- Code modifications to customized functionality can be made without having to modify the original application.
- Changes to the original application code can be made with minimal impact on the customizations.

Events can be used for different purposes, such as generating notifications when certain behavior occurs or the state of an entity changes, distributing information, and integrating with external systems and applications. For example, in the CRONUS International Ltd. demonstration database, events are used extensively for workflow and Microsoft Dynamics CRM integration.

How Events Work

The basic principal is that you program events in the application to run customized behavior when they occur. Events in Microsoft Dynamics NAV are modelled after Microsoft .NET Framework. There are three major participants involved in events: the event, a publisher and a subscriber.

An event is the declaration of the occurrence or change in the application. An event is declared by a C/AL function, which is referred to as an event publisher function. An event publisher function is comprised of a signature only and does not execute any code.

A publisher is the object that contains event publisher function that declares the event. The publisher exposes an event in the application to subscribers, essentially providing them with a hook-up point in the application.

Publishing an event does not actually do anything in the application apart from making the event available for subscription. The event must be raised for subscribers to respond. An event is raised by adding logic to the application that calls into the publisher to invoke the event (the event publisher function).

Partners or subsystems can then take advantage of the published event in their solutions. An ISV that delivers vertical solutions, and Microsoft itself, are the typical providers of published events.

There are three different event types: business, integration, and trigger events. For more information about each type, see Event Types. Business and integration type events must be explicitly declared and published, which means that you must create event publisher functions and add them to objects manually. On the other



hand, trigger events, which occur on table and page operations, are published and raised implicitly by the Microsoft Dynamics NAV runtime. Therefore, no coding is required to publish them.

A subscriber listens for and handles a published event. A subscriber is a C/AL function that subscribes to a specific event publisher function and includes the logic for handling the event. When an event is raised, the subscriber function is called and its code is run. A subscriber enables partners to hook into the core Microsoft Dynamics NAV application functionality without having to do traditional code modifications. Any Microsoft Dynamics NAV solution provider, which also includes Microsoft, can use event subscribers.

There can be multiple subscribers to a single event publisher function. However, a publisher has no knowledge of subscribers, if any. Subscribers can reside in different parts of the application than publishers.

Event types

Microsoft Dynamics NAV supports different types of events for different purposes. This topic describes the following event types:

Business Events

A business event is a custom event that is raised by C/AL code. It defines a formal contract that carries an implicit promise not to change in future releases. It is the expectation that business events are published by solution ISVs, including Microsoft.

Business events can be compared with publicly released APIs on which 3rd party solution providers develop integrations and additions. Therefore, the downstream cost of making changes to a business event implementation can be considerable for those who use the event in their applications. There may be some cases where changes are required; however, you should keep these to an absolute minimum.

Development considerations

A typical business event reflects changes in “state” with regards to a process. This makes them very well suited for workflow. An example of a business event could be when a sales order has been posted. It is important to note that business events should not be tied to the implementation-details, such as the tables or fields in which the data is stored. Preferably, the event publisher developer should be free to change the implementation, while still keeping the business event intact.

Business events should be documented with the solution, including the before-state and after-state of the events.

Integration Events

An integration event is also a custom event that is raised by C/AL code, like a business event, except that it does not carry the same promise of not changing, nor does it have the restriction not to expose implementation details.



The main purpose of integration events is to enable the integration of other solutions with Microsoft Dynamics NAV without having to perform traditional code modifications.

Development considerations

An integration event can be changed to a business event later. At which time, it must adhere to the same implied contract and commitment as any business event. It can also simply be designed-in hook points for external add-ons.

Global Events

Global events are predefined system events that are automatically raised by Codeunit 1 ApplicationManagement. Codeunit 1 includes several global function triggers, such as CompanyOpen, CompanyClose, and GetSystemIndicator. For most of these global function triggers, there are one or two global events: a before and after event. For example, there is an OnBeforeCompanyOpen event and an OnAfterCompanyOpen event. The global events are defined as integration events by local functions in codeunit 1.

Trigger Events

Unlike business and integration events which must be programmed, trigger events are predefined events. Trigger events are published by the runtime and they cannot be raised programmatically. There are two types of trigger events as described in this section: database trigger events and page trigger events.

[Event areas](#)

In addition to providing the capability to define your own events, the Dynamics NAV application comes with a set of events in different areas of the product. This document lists those events.

Posting

These events are for building solutions that extend the application posting without making changes to the posting codeunit themselves. Consider using these events instead of modifying the posting codeunits as it will improve the future upgrade of your solution.

Codeunit 1 Events

Many integration projects include changing and utilizing Codeunit 1 methods. These events are for building solution that extend the application without changing Codeunit 1 itself.

Workflow Events

The workflow events are all events that support the implementation of the out-of-the-box approval, notification, and process automation functionality. Using workflow events allows you to extend Microsoft workflows with custom functionality.

Workflow Extension Events



Use these events when you want to enable new types of workflows by making new workflow events and workflow responses. Using the library events below will ensure that your events show up in the Workflow Designer so that end users can take advantage of them.

CRM Integration Extensions

These events can be used to extend the CRM integration by adding custom mapping and more.

Trigger Events

Unlike business and integration events which must be programmed, trigger events are predefined events. Trigger events are published by the runtime and they cannot be raised programmatically. There are two types of trigger events: database trigger events and page trigger events.

Database Trigger Events

Trigger events are automatically raised by the system when it performs database operations on a table object, such as deleting, inserting, modifying, and renaming a record, as defined in a table. Trigger events are closely associated with the table triggers for database operations: OnDelete, OnInsert, OnModify, OnRename, and OnValidate (for fields). For each database operation, there is a "before" and "after" trigger event with a fixed signature.

Page Trigger Events

Page Trigger events are raised automatically by the system when it performs certain operations in a page object. Page trigger events are closely associated with the standard page triggers, such as OnOpenPage, OnClosePage, and OnAction.

In addition to providing the capability to define your own events, the Dynamics NAV application comes with a set of events in different areas of the product. This document lists those events.

Posting

Area	Event Name	Publisher Object
G/L	OnAfterCopyCustLedgerEntryFromGenJnlLine	Table 21
	OnAfterCopyGLEntryFromGenJnlLine	Table 17
	OnAfterCopyVendLedgerEntryFromGenJnlLine	Table 25
	OnAfterCheckGenJnlLine	Codeunit 11
	OnAfterInsertGlobalGLEntry	Codeunit 12
	OnAfterInitGLRegister	Codeunit 12
	OnBeforePostGenJnlLine	Codeunit 12
	OnBeforeInsertGLEntryBuffer	Codeunit 12
Item	OnAfterInitItemLedgEntry	Codeunit 22
	OnAfterInsertCorrItemLedgEntry	Codeunit 22
	OnAfterInsertCorrValueEntry	Codeunit 22
	OnAfterInsertItemLedgEntry	Codeunit 22
	OnAfterInsertValueEntry	Codeunit 22
	OnAfterCheckItemJnlLine	Codeunit 21
	OnAfterPostItemJnlLine	Codeunit 22
	OnBeforeInsertCorrItemLedgEntry	Codeunit 22
	OnBeforeInsertCorrValueEntry	Codeunit 22
	OnBeforeInsertTransferEntry	Codeunit 22
	OnBeforeInsertValueEntry	Codeunit 22
	OnBeforePostItemJnlLine	Codeunit 22
Purchase	OnAfterCalcPurchaseDiscount	Codeunit 70
	OnAfterPostPurchaseDoc	Codeunit 90
	OnAfterReleasePurchaseDoc	Codeunit 415
	OnAfterReopenPurchaseDoc	Codeunit 415
	OnBeforeCalcPurchaseDiscount	Codeunit 70
	OnBeforePostPurchaseDoc	Codeunit 90
	OnBeforePostCommitPurchaseDoc	Codeunit 90
	OnBeforeReleasePurchaseDoc	Codeunit 415
	OnBeforeReopenPurchaseDoc	Codeunit 415
Sales	OnAfterReleaseSalesDoc	Codeunit 414
	OnAfterReopenSalesDoc	Codeunit 414
	OnAfterCalcSalesDiscount	Codeunit 60
	OnAfterSalesDocPost	Codeunit 80
	OnBeforeReleaseSalesDoc	Codeunit 414
	OnBeforeReopenSalesDoc	Codeunit 414
	OnBeforeCalcSalesDiscount	Codeunit 60
	OnBeforeSalesDocPost	Codeunit 80

	OnBeforeSalesDocPostCommit	Codeunit 80
--	----------------------------	-------------

Codeunit 1 Events

Event Name	Publisher Object
OnAfterAutoFormatTranslate	Codeunit 1
OnAfterCaptionClassTranslate	Codeunit 1
OnAfterCompanyClose	Codeunit 1
OnAfterCompanyOpen	Codeunit 1
OnAfterFindPrinter	Codeunit 1
OnAfterGetApplicationVersion	Codeunit 1
OnAfterGetDatabaseTableTriggerSetup	Codeunit 1
OnAfterGetDefaultRoleCenter	Codeunit 1
OnAfterGetGlobalTableTriggerMask	Codeunit 1
OnAfterGetSystemIndicator	Codeunit 1
OnAfterMakeCodeFilter	Codeunit 1
OnAfterMakeDateFilter	Codeunit 1
OnAfterMakeDateText	Codeunit 1
OnAfterMakeDateTimeFilter	Codeunit 1
OnAfterMakeText	Codeunit 1
OnAfterMakeTextFilter	Codeunit 1
OnAfterMakeTimeFilter	Codeunit 1
OnAfterMakeTimeText	Codeunit 1
OnAfterOnDatabaseDelete	Codeunit 1
OnAfterOnDatabaseInsert	Codeunit 1
OnAfterOnDatabaseModify	Codeunit 1
OnAfterOnDatabaseRename	Codeunit 1
OnAfterOnGlobalDelete	Codeunit 1
OnAfterOnGlobalInsert	Codeunit 1
OnAfterOnGlobalModify	Codeunit 1
OnAfterOnGlobalRename	Codeunit 1
OnBeforeCompanyClose	Codeunit 1
OnBeforeCompanyOpen	Codeunit 1

Workflow Events

Area	Event Name	Publisher Object
Approvals	OnApproveApprovalRequest	Codeunit 1535
	OnCancelCustomerApprovalRequest	Codeunit 1535
	OnCancelGeneralJournalBatchApprovalRequest	Codeunit 1535
	OnCancelGeneralJournalLineApprovalRequest	Codeunit 1535
	OnCancelIncomingDocApprovalRequest	Codeunit 1535
	OnCancelItemApprovalRequest	Codeunit 1535
	OnCancelPurchaseApprovalRequest	Codeunit 1535
	OnCancelSalesApprovalRequest	Codeunit 1535
	OnCancelVendorApprovalRequest	Codeunit 1535
	OnDelegateApprovalRequest	Codeunit 1535
	OnRejectApprovalRequest	Codeunit 1535
	OnSendCustomerForApproval	Codeunit 1535
	OnSendGeneralJournalBatchForApproval	Codeunit 1535
	OnSendGeneralJournalLineForApproval	Codeunit 1535
	OnSendIncomingDocForApproval	Codeunit 1535
	OnSendItemForApproval	Codeunit 1535
	OnSendOverdueNotifications	Report 1509
	OnSendPurchaseDocForApproval	Codeunit 1535
	OnSendSalesDocForApproval	Codeunit 1535
OnSendVendorForApproval	Codeunit 1535	
Incoming Document	OnAfterReleaseIncomingDoc	Codeunit 132
OCR Integration	OnAfterCreateDocFromIncomingDocFail	Codeunit 132
	OnAfterCreateDocFromIncomingDocSuccess	Codeunit 132
	OnAfterIncomingDocReadyForOCR	Codeunit 133
	OnAfterIncomingDocReceivedFromOCR	Codeunit 133
	OnAfterIncomingDocSentToOCR	Codeunit 133
Purchase Document Posting	OnAfterPostPurchaseDoc	Codeunit 90
	OnAfterReleasePurchaseDoc	Codeunit 415
	OnAfterReopenPurchaseDoc	Codeunit 415
Sales Document Posting	OnAfterReleaseSalesDoc	Codeunit 414
	OnAfterReopenSalesDoc	Codeunit 414
	OnAfterSalesDocPost	Codeunit 80



Workflow Extension Events

Event Name	Publisher Object
OnAddWorkflowCategoriesToLibrary	Codeunit 1502
OnAddWorkflowEventPredecessorsToLibrary	Codeunit 1520
OnAddWorkflowEventsToLibrary	Codeunit 1520
OnAddWorkflowResponsePredecessorsToLibrary	Codeunit 1521
OnAddWorkflowResponsesToLibrary	Codeunit 1521
OnAddWorkflowTableRelationsToLibrary	Codeunit 1520
OnExecuteWorkflowResponse	Codeunit 1521

CRM Integration Extensions

Event Name	Publisher Object
OnAfterApplyRecordTemplate	Codeunit 5335
OnAfterInsertRecord	Codeunit 5335
OnAfterModifyRecord	Codeunit 5335
OnAfterTransferRecordFields	Codeunit 5335
OnBeforeApplyRecordTemplate	Codeunit 5335
OnBeforeInsertRecord	Codeunit 5335
OnBeforeModifyRecord	Codeunit 5335
OnBeforeTransferRecordFields	Codeunit 5335
OnFindUncoupledDestinationRecord	Codeunit 5335
OnQueryPostFilterIgnoreRecord	Codeunit 5340
OnRegisterServiceConnection	Table 1400



How to Implement Events

Implementing events in Microsoft Dynamics NAV consists of the following tasks:

Publish the event: For business and integration events, create and configure a function in an application object to be an event publisher function.

Raise the event: Add code that calls the event publisher function.

Subscribe to the event: At the consumer end, add one or more subscriber functions that subscribe to published events when they are raised.

Publishing an event

The first phase of implementing an event is publishing the event. Publishing an event exposes it in the application. This provides hook up points for subscribers to register to the event, and eventually handle the event if it is raised. An event is published by adding C/AL function that is specifically set up as an event publisher.

- Business and integration events require that you manually create an event publisher function for each event that you want to publish. An event publisher function declares the event in the application and makes it available for subscription; however, it does not raise the event. After an event is published, you can raise it in your application, as needed, from where event subscribers can react and handle the event.
- Trigger events, on the other hand, do not require that you create publisher functions. Trigger events are by predefined event publisher functions that are called automatically at runtime. This means that trigger events are readily available to subscribers by default.

Creating an Event Publisher Function to Publish Business and Integration Events

You create an event publisher function the same way you create any function in C/AL, except that there are specific properties that you set to make it an event publisher. Additionally, an event publisher function has the following requirements and restrictions that you must follow, otherwise you not able to compile your code changes:

- An event publisher function cannot include any code except comments.
- An event publisher function cannot have a return value, variables, or text constants.

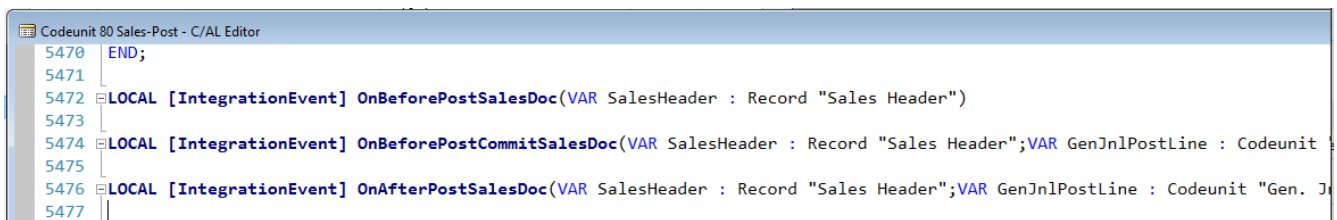
The following procedure provides an outline of the tasks that are involved in creating an event publisher function for declaring an event. The tasks are performed from the Microsoft Dynamics NAV Development Environment. For detailed step-by-step instructions for the tasks, see Walkthrough: Using Events in Microsoft Dynamics NAV.

To create an event publisher function

1. Decide where you want to include the event publisher function.
You can include an event publisher function in the C/AL code of any object type, such as codeunit, page, or table. You can create a new object or use an existing object.
2. Add a C/AL function to the object.
We recommend that you give the function a name that has the format On[Event], where [Event] is text that indicates what occurred, such as OnCancelCustomerApprovalRequest.
3. Set the function Event Property to Publisher.
4. Set the function EventType Property to Business or Integration.
5. If you want to make the event available to event subscribers that are defined in the other objects than publisher function object, set the Local Property to No.
6. If you want to make global functions in the object available to event subscribers, set the IncludeSender Property to Yes.
7. Add parameters to the function as needed.
You can include as many parameters of any type as necessary.
Make sure to expose enough information as parameters to enable subscriber functions to add value to the application. On the other hand, especially with business events, do not expose unnecessary parameters that may constrain you from changing or extending functionally in the future.

You can now add code to the application that raises the event by calling the publisher function. You can also create subscriber functions that handle the event when it is raised.

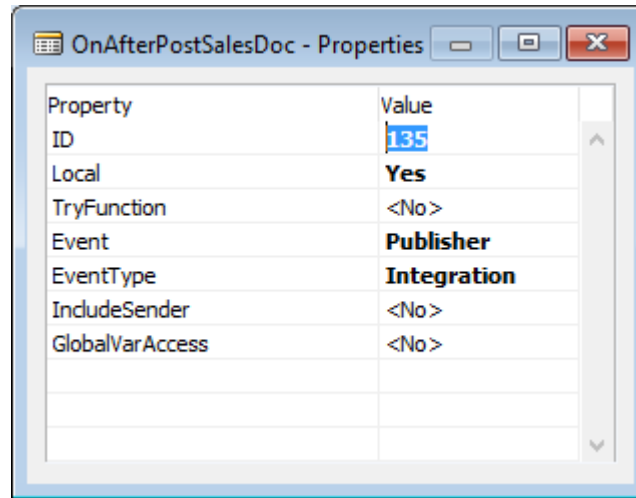
Looking at the existing publications in codeunit 80 there are three predefined publications we can use:



```
Codeunit 80 Sales-Post - C/AL Editor
5470 END;
5471
5472 LOCAL [IntegrationEvent] OnBeforePostSalesDoc(VAR SalesHeader : Record "Sales Header")
5473
5474 LOCAL [IntegrationEvent] OnBeforePostCommitSalesDoc(VAR SalesHeader : Record "Sales Header";VAR GenJnlPostLine : Codeunit
5475
5476 LOCAL [IntegrationEvent] OnAfterPostSalesDoc(VAR SalesHeader : Record "Sales Header";VAR GenJnlPostLine : Codeunit "Gen. J
5477
```

The OnAfterPostSalesDoc publication has been defined as a local function and can therefore only be raised from inside this object.

The other properties of the function has been set to:



It is a Publisher event and the type is an Integration.

Creating new Publisher events can be stored in a separate codeunit and it can be raised just by addressing the the event with Codeunit.PublisherEventName. The only difference is that, the Publisher events local property my be set to No.

Raising an event

After an event has been published by an event publisher function, you can modify the application to raise the event where it is needed. Subscribers of an event will not react on the event until it is raised in the application. To raise an event, you add logic in C/AL code of the application to call the event publisher function that declares the event. The procedure for calling the event publisher function is the same as calling any other function in C/AL.

When the code that calls the event publisher function is run, then all event subscriber functions that subscribe to the event are run. If there are multiple subscribers, then each event subscriber function is run one after another. The order in which the event subscribers run is random and cannot it cannot be specified. If there are no subscribers to the published event, then the line of code that calls the event publisher function is ignored and not executed.

In the case of the OnAfterPostSalesDoc event, it is raised in the end of the main function after everything else has been executed:

```
Codeunit 80 Sales-Post - C/AL Editor
1252 UpdateItemAnalysisView.UpdateAll(0,TRUE);
1253 END;
1254
1255 // Balancing account - online payment
1256 IF ("Bal. Account No." <> '') AND IsOnlinePayment(SalesHeader) AND Invoice THEN
1257   PostBalanceEntry(
1258     TransactionLogEntryNo,SalesHeader,TotalSalesLine,TotalSalesLineLCY,
1259     GenJnlLineDocType,GenJnlLineDocNo,GenJnlLineExtDocNo,SrcCode);
1260
1261 CRMIntegrationManagement.AddPostedSalesDocumentToCRMAccountWall(SalesHeader);
1262 CRMIntegrationManagement.SetCRMSalesOrderStatusToInvoiced(SalesHeader);
1263
1264 OnAfterPostSalesDoc(Rec,GenJnlPostLine,SalesShptHeader."No.",ReturnRcptHeader."No.",SalesInvHeader."No.",
1265   SalesCrMemoHeader."No.");
1266
1267 SetPostingDate(NewReplacePostingDate : Boolean;NewReplaceDocumentDate : Boolean;NewPostingDate : Date)
1268   PostingDateExists := TRUE;
```

This means that everytime a sales order is posted Dynamics NAV will look for any codeunits that subscribe to this publication.

Subscribing to Events

To handle events, you design event subscribers. Event subscribers determine what actions to take in response to an event that has been raised. An event subscriber is a C/AL function that subscribes to, or listens for, a specific event that is declared by an event publisher function. The event subscriber includes code that defines the business logic to handle the event. When the published event is raised, the event subscriber is called and its code is run.

Subscribing to an event tells the runtime that the subscriber function must be called whenever the publisher function is run, either by code (as with business and integration events) or by the system (as with trigger events). The runtime establishes the link between an event raised by the publisher and its subscribers by looking for event subscriber functions.

There can be multiple subscribers to the same event from various locations in the application code. When an event is raised, the subscriber functions are run one at a time in random order. You cannot specify the order in which the subscriber functions are called.

Be aware that changing the state may not only impact the publishing code but other subscribers as well.

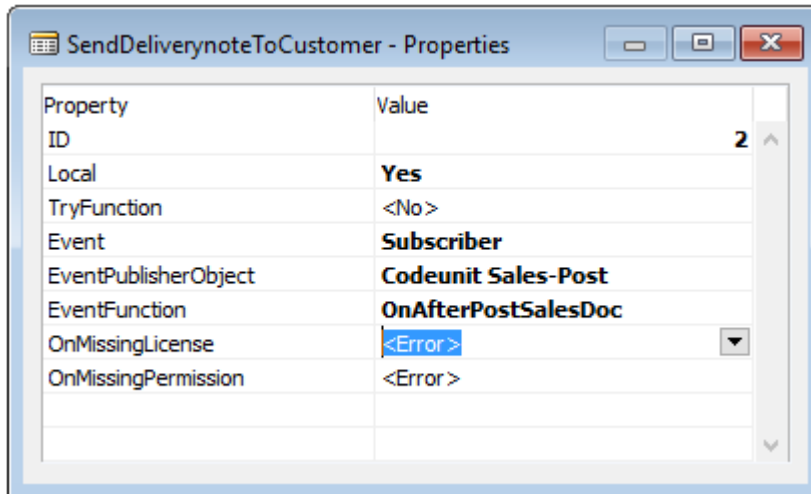
Subscriber functions cannot access the sender and or access global variables.

Example of an Event Subscriber Function

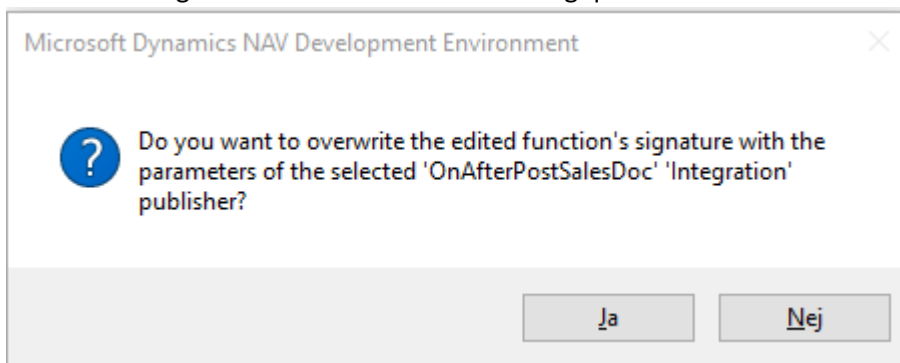
You create an event subscriber function just like other functions except that you specify properties that set up the subscription to an event publisher. The procedure is slightly different for database and page trigger events than business and integration events because business and integration events are raised by event publisher functions in application code. Trigger events are predefined system events that are raised automatically on tables and pages.

In this example we can make a subscriber function to send an email to the customer with the delivery note whenever a sales order is posted. The process is as follows:

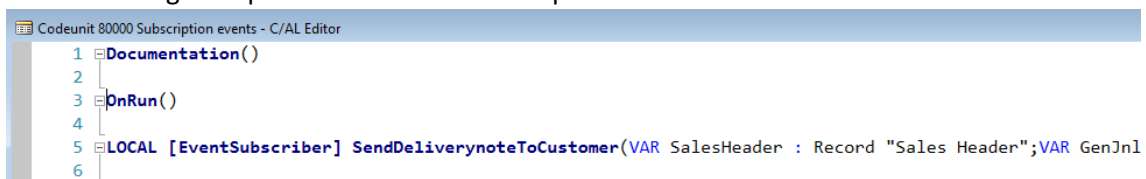
- 1) Create a new codeunit to hold the subscriber event
- 2) Create a function
- 3) Change the properties



- 4) When selecting the EventFunction the following question is asked



- 5) This will change the parameters to match the publication event



6) Now add the code that you want executed

```

Codeunit 80000 Subscription events - C/AL Editor
1 Documentation()
2
3 OnRun()
4
5 LOCAL [EventSubscriber] SendDeliverynoteToCustomer(VAR SalesHeader : Record "Sales Header";VAR GenJnl)
6 IF SalesHeader.Ship THEN BEGIN
7   IF SalesShptHeader.GET(SalesShptHdrNo) THEN BEGIN
8     SalesShptHeader.SETRECFILTER;
9     DeliveryNote.SETTABLEVIEW(SalesShptHeader);
10    IF DeliveryNote.SAVEASPDF(FileNameTxt) THEN BEGIN
11      CompanyInfo.GET;
12      SalesPerson.GET(SalesHeader."Salesperson Code");
13      Customer.GET(SalesHeader."Sell-to Customer No.");
14      Mail.CreateMessage(CompanyInfo.Name,
15        CompanyInfo."E-Mail",
16        Customer."E-Mail",
17        STRSUBSTNO(DeliveryNoteTxt,SalesShptHeader."No."),
18        STRSUBSTNO(BodyTxt,SalesHeader."Sell-to Contact",SalesPerson.Name),FALSE);
19      Mail.AddAttachment(DeliveryNoteTxt,FileNameTxt);
20      Mail.Send;
21      IF EXISTS(FileNameTxt) THEN
22        ERASE(FileNameTxt);
23    END;
24  END;
25 END;
26

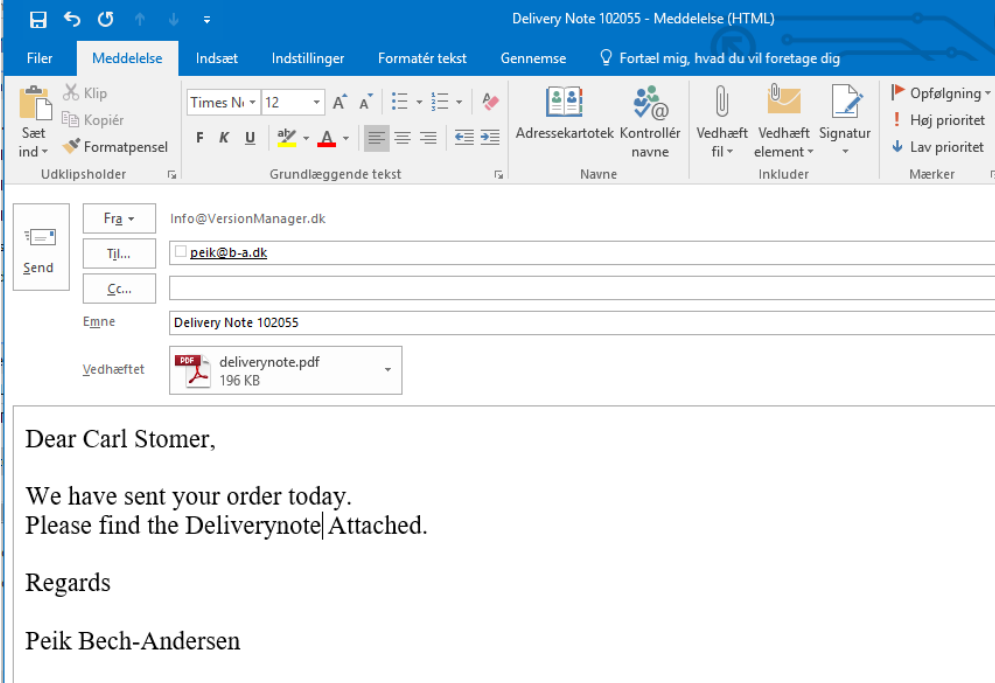
```

7) Test the subscription event

1012 · C. U. Stomer

General									
No.:	1012	Document Date:	31-12-2017						
Sell-to Customer No.:	C00010	Requested Delivery Date:							
Sell-to Customer Name:	C. U. Stomer	External Document No.:							
Sell-to City:	Miami	Salesperson Code:	PS						
Posting Date:	31-12-2017	Status:	Open						
Order Date:	31-12-2017								
Show more fields									
Lines									
Line	Functions	Order	New	Find	Filter	Clear Filter			
Type	No.	Description	Quantity	Unit of Measur...	Unit Price Excl. VAT	Line Amount Excl. VAT	Deferral Code	Line Discount %	
Item	1000	Bicycle	1	PCS	4.000,00	4.000,00			

Ship the sales order and the mail will pop up:



The screenshot shows an Outlook email window titled "Delivery Note 102055 - Meddelelse (HTML)". The interface includes a ribbon with tabs for "Filer", "Meddelelse", "Indsæt", "Indstillinger", "Formatér tekst", "Gennemse", and "Fortæl mig, hvad du vil foretage dig". The "Meddelelse" tab is active, showing a rich text editor with various formatting options like font size, bold, italic, underline, and color. The email content is as follows:

From: Info@VersionManager.dk
To: peik@b-a.dk
Subject: Delivery Note 102055
Attachment: [deliverynote.pdf](#) (196 KB)

Dear Carl Stomer,

We have sent your order today.
Please find the Deliverynote|Attached.

Regards

Peik Bech-Andersen

Be aware that any errors raised in the subscription event WILL be shown as normal, but it will not perform a roll-back as usual. This means that the order will be posted regardless of any errors raised in the subscription code.

Extensions

What are extensions?

How Extensions work

Lifecycle of an extension

Windows PowerShell Cmdlets

Extension Packages

Develop an extension

Building the extension package

To sign an extension package

Publishing and installing extensions

Supported functionality in extensions

Extension example

This chapter is a compilation of available Microsoft material. More information can be found at [https://msdn.microsoft.com/en-us/library/mt574417\(v=nav.90\).aspx](https://msdn.microsoft.com/en-us/library/mt574417(v=nav.90).aspx)





What are extensions?

You can extend and customize a Microsoft Dynamics NAV deployment without modifying the original application objects. With extension packages, you install, upgrade, and uninstall functionality in on-premises deployments or for select tenants in a multitenant deployment. Customers can easily add or remove horizontal or customized functionality to their solution that upgrade much easier than past solutions.

In Microsoft Dynamics NAV 2016, you can create extensions to customize pages and tables. The main difference from classical development is that source code modifications are not allowed. Instead, you use C/AL events to extend and customize objects.

Extensions are delivered as .navx package files. A .navx package contains the various artifacts that deliver the new functionality to the Microsoft Dynamics NAV deployment as well as a manifest that specifies the name, publisher, version, and other attributes of the extension. You manage .navx packages with a series of Windows PowerShell cmdlets that are available in the Microsoft Dynamics NAV 2016 Administration Shell. There are also cmdlets available to ISVs and developers in the Microsoft Dynamics NAV 2016 Development Shell that help create packages.

How extensions work

Extensions are in the simplest terms the runtime application of objects and object deltas for a specific combination of an extension package and a tenant. When an extension is published to a Microsoft Dynamics NAV deployment, it compiles the objects in it against the current application database. Then, when the extension installs for a tenant, it stores the association and builds the relevant database schema. At runtime, Microsoft Dynamics NAV simply loads the associated objects for that extension and tenant.

You can publish multiple extensions to a Microsoft Dynamics NAV deployment and, in multitenant deployments, install any combination of published extensions for each tenant. For example, consider a scenario with a multitenant deployment with extensions A, B, C, and D published to it. Each tenant can have their own unique combination of extensions. So tenants 2, 3, and 4 can have the same extensions (extensions B and C) while tenants 1 and 5 only have one extension each, extension A and D, respectively. This provides for a great degree of customer functionality choice while at the same time maximizes the server hardware and administration workload.

In most cases, two extension packages can coexist and work independently of each other; however there is the possibility that two apps will try to modify the same object properties. In those cases, if the conflict cannot be resolved, the installation of the conflicting extension fails.

Using Extensions

Extensions as a concept are designed for horizontal add-ons that are additive by nature. While not limited to that premise, following that guideline will increase the ability of your extension to coexist with other extensions. It will also ensure seamless upgrade across core application versions as well as versions of the extension.



Code modifications are not allowed in extension packages. This is intended to remove the code-merge task during upgrade, and to increase the probability that different extensions can coexist.

Changes to a Microsoft Dynamics NAV deployment by an extension are treated differently than if you had manually added them to your code base. For more information, see Upgrade.

In Microsoft Dynamics NAV 2016, not all core concepts are supported. We have focused on establishing a strong foundation for extensions that includes the most important and most commonly used elements of an add-on, such as pages (modifications and full objects), tables (modifications and full objects), code units (full objects), and menu suites (full objects). For a full list, see the Extension Packages Capability Support Matrix.

Licensing Considerations

Licensing is the same for functionality that is made available in an extension package as in an .fob file. You must have the relevant license to create and export objects as TXT files. Similarly, your consumers must also have the appropriate licensing to use your extension.

Lifecycle of an Extension

By design, extension packages are intended to be more easily upgraded, so an extension is based on the concept of deltas that were introduced in the code upgrade process with Application Merge Utilities in Microsoft Dynamics NAV 2015. With this tool, you can create delta files for metadata modifications, which then can be used to merge into a target environment. When you publish an extension, this merging concept is taken to the next level where a similar delta is merged and compiled against the existing application. The difference is that the merged object is loaded at runtime as opposed to being embedded in the core source object. Doing this also permits this merge to be effectively reversed by simply uninstalling the extension package, which removes that delta object from the deployment.

Development

As a developer, you maintain your extension as a codebase that is separate from the original, and then export the modified objects as standard object .TXT files. Next, you run the application merge utilities to create delta files and then build the package with new Windows PowerShell cmdlets that are introduced in Microsoft Dynamics NAV 2016. To update an extension, you simply repeat that process, incrementing the extension version number.

Packaging

An integral part of development is the final building of the package. This includes providing the extension with metadata, such as name, publisher, and version, in a manifest, and packaging that with the application elements of the extension, such as DELTA files and permission set export files. The end result of that process is a package of type .NAVX .

Deployment

You administrate extensions by using a set of Windows PowerShell cmdlets to publish extensions to a Microsoft Dynamics NAV deployment, which makes it available for tenants to selectively install and use. You can also uninstall extensions for a tenant, or remove an extension from the Microsoft Dynamics NAV



deployment. Upgrade scenarios, either for a specific extension or the application for this Microsoft Dynamics NAV deployment, can be accomplished with less friction as extensions, and their functionality are self-contained and do not contain alterations to C/AL code, removing the need to perform a manual merge.

You can publish multiple versions of the same extension to a Microsoft Dynamics NAV deployment. However, you can only have one version of the extension installed for each tenant. In a multitenant deployment, one tenant can run a different version of an extension than other tenants.

Upgrade

Changes to a Microsoft Dynamics NAV deployment by an extension are treated differently than if you had manually added them to your code base. UI modifications are basically just the differences from the base application code, and are applied at runtime. Because you cannot modify any existing code in an extension package, changes in business logic must be done through the use of events and new codeunits. As it is, you can upgrade your extension independently of the upgrade process of the Microsoft Dynamics NAV deployment as a whole.

In short, to upgrade an extension, you uninstall the old package and then install a new version of the package. However, when an extension is uninstalled, all data for the extension is archived into a set of special tables. When you change tables or add new tables, you must write upgrade code that determines what to do with the archived data when the extension is installed again.

Move Tenants from One Server to Another

In multitenant deployments, moving a tenant from one server to another requires some evaluation with regards to your extension packages. The data for any extension is stored within the schema for each company in the tenant database so that will move without any extra steps. The tenant database also stores information about which extensions are installed for that tenant. Moving a tenant requires an examination of the destination server to validate that all extensions for the tenant are published on the on the destination server.

As part of your preparations to move a tenant, review all installed extension packages for the tenant to make sure the same extensions are available on the destination server. You can use the Get-NAVAppInfo cmdlet to accomplish this. If the apps are available then you can simply unmount the tenant and move it, as part of the mount tenant process all installed extensions for the tenant being moved will be re-installed. Please note that the applications that the two servers use must of course also be identical for this process to succeed.

If your destination server does not have the required extensions, you can alternately uninstall the extension packages before you dismount to allow the mounting of the tenant on the destination server. The data for those extensions will be kept in the archived tables until you publish and reinstall that extension, or you choose to no longer use the extension, at which time you will need to discard the data by removing the archived tables in the tenant database.

System Tables and Virtual Tables

Extension packages are installed to and operate on a per-tenant basis. To accomplish this, the objects that are used at runtime are dynamically constructed with any extension deltas that are present for the base object. The end result is a virtual merge of the base plus the delta for a given extension. Multiple deltas from multiple



extension packages can also be applied in this process. In Microsoft Dynamics NAV 2016, new system tables are added to support extensions. We recommend that you only access the system tables using the cmdlets in the Microsoft Dynamics NAV 2016 Administration Shell and Microsoft Dynamics NAV 2016 Development Shell. They are listed here for your information only:

- NAV App Object Metadata – Stores the metadata deltas for an extension.
- NAV App Tenant App – Contains all extensions that are installed to tenants and part of the application database. This table is synchronized with NAV App Installed App in the tenant databases and is used to drive runtime decisions about tenant/app relationships. It is also watched for changes by each Microsoft Dynamics NAV Server instance to support on-the-fly app switching.
- NAV App Data Archive – Points to tables created to archive data for extensions that have been uninstalled. Used to provide access to the data when the extension is reinstalled during an upgrade.
- NAV App Installed App – Contains information about extensions that are installed in this tenant. This lives in the tenant database and is written/read on mount/unmount to make sure tenants keep track of their apps even when detached from an application database.
- NAV App - Contains all extensions that are published to the system. This is part of the application database.
- NAV App Dependencies – List of dependencies for individual extensions that are published to the system.
- NAV App Capabilities – List of capabilities for each extension that is published to the system. Pulled from the manifest at publication time.
- NAV App Object Prerequisites – List of prerequisite objects for individual extensions that are published to the system.
- Tenant Permission Set - Defines the mapping between extensions and permission sets.
- Tenant Permission – Contains permissions for objects as defined by permission sets in extension packages.

The objects from extensions are also included in the following virtual tables

AllObj, AllObjWithCaption includes new objects from all installed extensions based on the current session/tenant.

AllObj, AllObjWithCaption includes the App Package ID field (empty GUID for base objects, package id GUID if object comes from an extension)

[Windows PowerShell Cmdlets](#)

To help you build, package, publish, install, and manage extensions, Microsoft Dynamics NAV includes a number of Windows PowerShell cmdlets as described in the following sections.

Development

The Microsoft Dynamics NAV 2016 Development Shell includes the following new cmdlets:

Area	PowerShell Cmdlets
Manifest	New-NAVAppManifest Set-NAVAppManifest Get-NAVAppManifest New-NAVAppManifestFile
Permissions	Export- NAVAppPermissionSet
Packaging	New-NAVAppPackage

Administration

The Microsoft Dynamics NAV 2016 Administration Shell includes the following new cmdlets:

Area	PowerShell Cmdlets
Publishing	Publish-NAVApp Unpublish-NAVApp
Installing	Install-NAVApp Uninstall-NAVApp
Managing	Get-NAVAppInfo Get-NAVAppTenant Repair-NAVApp

Extension Packages

You deploy your Microsoft Dynamics NAV extensions as packages.

A package file can contain a number of different artifacts that are needed for the extension, such as DELTA files and full application object .TXT files. The package also contains a manifest with metadata such as name, version, publisher and, most importantly, a globally unique identifier, the extension ID. Microsoft Dynamics NAV uses the extension ID to uniquely identify the extension. This ID must be maintained consistently across versions of an extension in order to upgrade properly. The Windows PowerShell cmdlets that you use to create the manifest generate an ID for you, but you can also specify an ID.

Extension Manifests

Use the `New-NAVAppManifestFile` cmdlet to write your manifest to disk so that the App ID is preserved. Then, you can use the modification cmdlet, `Set-NAVAppManifest`, to change attributes such as the version number.

The following code snippet illustrates how you can generate a manifest file.

```
New-NAVAppManifest -Name 'My Extension' -Publisher 'Proseware' -Version '1.0.0.0' |
New-NAVAppManifestFile -Path 'C:\MyExtensionManifest.xml'
```

The following code example illustrates a manifest for an extension.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="http://schemas.microsoft.com/navx/2015/manifest">
<App Id="6147E5EF-197E-43CF-B4A2-168AAE537A0B"
Name="Microsoft Dynamics NAV Customer Loyalty"
Publisher="Microsoft Corporation"
Description="Customer Loyalty Extension Package for Microsoft Dynamics NAV"
Version="1.0.23.0"
CompatibilityId="1.0.0.0" />
<Capabilities>
<Capability Name="schemaChanges" />
</Capabilities>
<Prerequisites>
<Objects>
<Object Type="CodeUnit" Id="1600"/>
</Objects>
</Prerequisites>
<Dependencies>
<Dependency Id="9DE57280-CC68-4612-BB26-8C8F5CF34A37"
Name="Microsoft Dynamics NAV Service Integration"
Publisher="Microsoft Corporation"
MinVersion="1.3.1.0"
CompatibilityId="1.1.0.0" />
</Dependencies>
</Package>
```

At the root of the manifest is the `Package` element. The following sections describe the other elements and settings.

App

The `App` element describes your extension to the system and users. The element has the following attributes.

Attribute	Required	Type	Description
Id	Yes	GUID	An identifier that uniquely identifies the extension. This value should remain constant for the life of the application, meaning that the same identifier should be used for each new version of the extension. A unique identifier will be generated if a value is not provided.
Name	Yes	String	The name of the extension. String length of 250 characters.
Publisher	Yes	String	The publisher of the extension. String length of 250 characters.
Description	No	String	The description of the extension. String length of 2048 characters.
Version	Yes	String	Version string in the format of Major.Minor.Build.Revision. This value should be incremented for each published package.
CompatibilityId	Yes	String	Version string in the format of Major.Minor.Build.Revision. This value defines the compatibility ID of the extension.



Capabilities

The Capabilities element contains a collection of Capability elements that describe what the extension impacts in terms of the hosting Microsoft Dynamics NAV deployment. The Capability element can be used to determine if an extension is allowed to be installed.

Capabilities cannot be set manually as they are added by the packaging process. Microsoft Dynamics NAV 2016 does not provide any functionality for analyzing or otherwise reacting to capabilities. Capabilities are reserved for future use.

Prerequisites

The Prerequisites element contains a collection of Object elements that define what must be presently available in the hosting Microsoft Dynamics NAV deployment in order for this extension to function. It uses specific identifiable object IDs that must be available in the system. If a prerequisite is not found, the extension cannot install.

Attribute	Required	Type	Description
Type	Yes	Type	The type of the object. The following objects are available: TableData Table Report CodeUnit XmlPort MenuSuite Page Query
Id	Yes	Int32	The object ID.

Dependencies

The Dependencies element contains a collection of Dependency elements that define identifies other extensions that this extension has a specific dependency on. Dependencies are added to the manifest when you provide a path to a .navx file in the –Dependencies parameter of the New-NAVAppManifest cmdlet.

Attribute	Required	Type	Description
Id	Yes	GUID	The unique identifier of the extension that this extension depends on.
Name	Yes	String	The name of the extension.
Publisher	Yes	String	The publisher of the extension.
MinVersion	Yes	String	Version string in the format of Major.Minor.Build.Revision. This is the minimum version of the extension that must be present in conjunction with the specified Compatibility ID.

CompatibilityId	Yes	String	Version string in the format of Major.Minor.Build.Revision. This value defines the compatibility ID of the extension.
------------------------	-----	--------	---

Develop an Extension

You can build extension packages that add functionality to a Microsoft Dynamics NAV deployment. Unlike the familiar development and deployment of Microsoft Dynamics NAV functionality, building an extension relies on the exported version of an application to .TXT files. You can export the application from the development environment, or use the Windows PowerShell function that is available in the Microsoft Dynamics NAV 2016 Development Shell, `Export-NAVApplicationObject`.

It is recommended that you create a folder structure that can be leveraged by the cmdlets that you use when you build the extension package. That structure should contain folders for the ORIGINAL object files, MODIFIED object files, and DELTA files. These names match those used as parameters in the application merge utilities.

To create an extension

- 1) Establish the BASE as TXT files.

The foundation for your extension is the exported .txt files of the objects you are modifying. You can export just the objects that you want to modify, or you can export the entire Microsoft Dynamics NAV application. In the Microsoft Dynamics NAV 2016 Development Shell, the `Export-NAVApplicationObject` function can automate this process or you can use the export functionality in the development environment. The following example uses this function to export objects to establish the base for the planned modifications.

```
Export-NAVApplicationObject -Path ORIGINAL -DatabaseName MyDatabase -  
DatabaseServer MyDatabaseServer
```

Objects must be exported as .TXT files. You cannot build an extension based on a .FOB file.

If you use a source control system, you may want to pull the base .TXT files from there.

- 2) Create functionality in the development environment.
 - a. Use the development environment as usual to create new objects or modify ones to the extent your license allows you. Keep in mind the following rules:
 - DO NOT make C/AL code modifications
 - DO use subscribing to events to execute code.
 - DO NOT create new or modified XMLPorts, Queries, or Report.
 - DO NOT change restricted page and table properties.

In order to get an easy upgrade experience for your extensions, you cannot modify code the way you do in the traditional customization process. Instead, you extend Microsoft Dynamics NAV functionality by subscribing to programming events that are raised either explicitly in code, or implicitly by the platform.

- b. Add upgrade code for new or modified tables.
The purpose of the upgrade code is to handle table data that is related to the extension when the extension is uninstalled and reinstalled. You add the code to two global functions, one named **OnNavAppUpgradePerDatabase** and the other named **OnNavAppUpgradePerCompany**.
- c. Test your application with the extension added.

3) Export your changed and added application objects to .TXT files.

Export all objects that you added or modified to .TXT files. Use the same export function from step 1 or manually export within the development environment. They must also be exported as .TXT files and should be placed in a separate directory so that the packaging process can be pointed at them.

```
Export-NAVApplicationObject -Path MODIFIED -DatabaseName MyDatabase -  
DatabaseServer MyDatabaseServer
```

4) Create DELTA files using the Microsoft Dynamics NAV 2016 Development Shell cmdlets.

Extension packages are based on application object deltas. Again, you use the application merge utilities in the Microsoft Dynamics NAV 2016 Development Shell to distill the changes in the form of application object differences that are stored in DELTA files. Creating an extension uses many of the same concepts and tools as you know from application object deltas. You use the `Compare-NAVApplicationObject` cmdlet to create these delta files.

```
Compare-NAVApplicationObject -OriginalPath ORIGINAL -ModifiedPath MODIFIED -  
DeltaPath DELTA
```

Your delta files must be one-to-one with the objects you have added or modified. You cannot include a single merged delta file. If you output your export file as a single file use the `Split-NAVApplicationObjectFile` cmdlet to create the individual files.

5) Build the extension package.



Building the extension package

After having developed a Microsoft Dynamics NAV extension, the next step is to wrap your new .TXT and .DELTA files into a .navx file, the packaging format for extensions. The package includes the application objects and metadata that describes your extension, such as name and version.

To create the extension manifest

The extension manifest describes characteristics about your extension. All characteristics have a parameter in the Windows PowerShell cmdlet that you use to create the manifest, `New-NAVAppManifest`. The following table describes the data in the manifest:

Data	Description
Name	Specifies the name of the extension.
Publisher	Specifies the publisher of the extension, such as your company name.
Version	Specifies the version of the extension. The version is a string in the format of Major.Minor.Build.Revision, with a default value of 1.0.0.0. You must increment the value for each new version of the extension that you publish.
Description	Specifies the description for the extension.
Id	Specifies the unique identifier for the extension. A unique identifier will be generated if a value is not provided. The same unique identifier should be used for each new version of the extension.
CompatibilityId	Specifies the compatibility ID of the extension. The compatibility ID is a version string in the format of Major.Minor.Build.Revision, with a default value of 1.0.0.0. The value is used to indicate whether there are compatibility related code changes between different versions of the extension. If a new version of the extension does not break compatibility, leave the compatibility ID the same as the previous version.
Dependencies	Specifies the path to a package file (.navx) for another extension that this extension depends on. Use a comma (,) to separate the paths to multiple .navx files., such as in the following example: C:\Proseware\SmartAppBase.navx, C:\Proseware\ProsewareBase.navx
Prerequisites	Specifies the objects that must exist in order to deploy the extension to a Microsoft Dynamics NAV Server instance. The prerequisites is a string in the format of type=ID, where type can be any object type such as Table, CodeUnit, or Page. Use a comma (,) to separate the prerequisites, such as in the following example: Table=397, CodeUnit=78.

The `New-NAVAppManifest` cmdlet creates an in-memory Manifest object.

```
New-NAVAppManifest -Name "Proseware SmartStuff" -Publisher "Proseware, Inc."  
-Version "1.5.0.12" | New-NAVAppManifestFile -Path '.\Manifest-Proseware  
SmartStuff.xml' -Force
```

You can either persist this object to a file and then check it in to source control by using `New-NAVAppManifestFile`, or you can pass it directly to as described in the next step.

```
Get-NAVAppManifest -Path '.\Manifest-Proseware SmartStuff.xml' | New-  
NAVAppPackage -Path MyExtension.navx -SourcePath DELTA
```



You have packaged your extension so it is ready to be published and installed on a target server.

The packaging process adds a description of the extension to the manifest, such as whether it changes pages or adds tables. While not explicitly being enforced currently, this can be used to determine whether to install an extension, or not. Use `Get-NAVAppManifest -Path` to see capabilities.

Finally, you can choose to get your extension package signed.

To sign an extension package

To help validate the authenticity of an extension package (the .navx file), we recommend that you have it signed. Code signing is a common practice for many applications.

- 1) To sign an extension package, you need a computer that has the following:
 - a. A code signing tool, such as SignTool or CodeSign.
SignTool is part of the Windows Software Development Toolkit.
For more information, see [SignTool](#).
 - b. Microsoft Dynamics NAV 2016 or later.
- 2) Obtain a certificate that is enabled for the code signing purpose. You can have certificate as a file or installed in the certificate store of the computer.
In production, we recommend that you use a certificate from a third party certificate authority. However, for testing purposes, it is acceptable to create a self-signed certificate, which, for example, you can create by using the [New-SelfSignedCertificate cmdlet](#) in Windows PowerShell or [MakeCert](#).

The following example uses the **New-SelfSignedCertificate** cmdlet to create a new self-signed certificate:

```
New-SelfSignedCertificate -Type CodeSigningCert -Subject "CN=ProsewareTest"
```

The following example uses the MakeCert tool to create a new self-signed certificate:

```
makecert -sk myNewKey -n "CN=Prosewaretest" -r -ss my
```

- 3) It is optional but we recommend that you use a time stamp when signing the .navx file.
A time stamp allows the .navx file signature to be verifiable even after the certificate that is used for the signature has expired.
For more information, see [Time Stamping Authenticode Signatures](#).
- 4) Sign the .navx file by using your signing tool.
See the following examples for signing by using SignTool.



Example

The following example signs the Proseware.navx file with the certificate in the password-protected MyCert.pfx file.

```
SignTool sign /f MyCert.pfx /p MyPassword "C:\NAV\Proseware.navx"
```

The following example signs the Proseware.navx file with the certificate in the password-protected MyCert.pfx file and a time stamp.

```
SignTool sign /f MyCert.pfx /p MyPassword /t  
http://timestamp.verisign.com/scripts/timestamp.dll "C:\NAV\Proseware.navx"
```

The following example signs the Proseware.navx file with the certificate in the store called My with a subject name of Prosewaretest.

```
SignTool sign /n Prosewaretest "C:\NAV\Extension\Proseware.navx"
```

Publishing and Installing an Extension

To make your extension available to users, the package must be published to a specific Microsoft Dynamics NAV Server instance. The extension can be installed for one or more tenants.

To publish or remove an extension

Publishing an extension

In the Microsoft Dynamics NAV 2016 Administration Shell, use the Publish-NAVApp cmdlet. The cmdlet takes as parameters the server you want to install to and the .navx package file that contains the extension. The following example publishes the extension MyExtension to the YourDynamicsNAVServer instance.

```
Publish-NAVApp -ServerInstance YourDynamicsNAVServer -Path MyExtension.navx
```

Publish does more than just update internal tables. It also compiles the components of the extension behind-the-scenes and builds the necessary metadata objects that are used at runtime.

You can get an overview of the published extensions and their state using the Get-NAVAppInfo cmdlet. If no tenants have a specific extension installed, you can completely remove it using the Unpublish-NAVApp cmdlet.

Removing an extension

In the Microsoft Dynamics NAV 2016 Administration Shell, use the Unpublish-NAVApp cmdlet. The cmdlet takes as parameters the server you want to remove the extension from, and the name of the extension. The following example removes the extension MyExtension from the YourDynamicsNAVServer instance.



```
Unpublish-NAVApp -ServerInstance YourDynamicsNAVServer -Path MyExtension
```

Once an app has been published, it must be made available for any tenant that wishes to use it.

To install an extension

In the , use the Install-NAVApp cmdlet. The following example installs the MyExtension for Tenant1 and Tenant3. In single-tenant deployments, you either specify default as the tenant ID, or you omit the –Tenant parameter.

```
Install-NAVApp -ServerInstance YourDynamicsNAVServer -Name "My Extension" -  
Tenant Tenant1, Tenant3
```

se Get-NAVAppInfo –Tenant command to get an overview of the extensions for that tenant, use the Get-NAVAppTenant cmdlet to get all tenants that have installed a specified extension, and uninstall an extension using the Uninstall-NAVApp cmdlet.

When you uninstall an extension that includes tables and fields, this impacts the database schema and any data that the tables and fields contain.



Supported functionality in Extension Packages

An extension package can contain both new and modified existing objects. In Microsoft Dynamics NAV 2016, the following object types can be added and included in an extension:

- Pages
- Tables
- MenuSuites
- Codeunits

You can also modify existing objects of the following types as permitted by your license:

- Pages
- Tables
- Action Items

You cannot add or modify any other object types in this version. Also, you cannot delete any existing objects.

You cannot modify any existing C/AL code, including code in codeunits and in triggers on existing objects. If you want to modify existing code, use the new C/AL eventing model. This restriction is only on existing code and objects. New pages, tables, and so on, can contain C/AL code as it is considered part of the new object.

Restricted Properties

There are restrictions on certain property changes for modified objects. The following sections list the properties you cannot change. The cmdlets that create and install packages will halt with errors if any of these properties are changed in your extension.

Restricted Properties on Existing Page Modifications

You cannot change the values for the following properties for existing Microsoft Dynamics NAV pages in an extension.

AccessByPermission	FieldClass	PastelsValid
AssistEdit	GroupType	Permissions
AutoSplitKey	ID	PopulateAllFields
CardPageID	InsertAllowed	RefreshOnActivate
CharAllowed	LinkedObject	SourceExpr
ContainerType	Lookup	SourceTable
ControlAddIn	LookupPageID	SourceTableTemporary
Data Type	MaxValue	SourceTableView
DataLength	MinValue	SubType
DateFormula	ModifyAllowed	SystemPartID
DelayedInsert	MultipleNewLines	TableRelation
DeleteAllowed	Name	TableType
DrillDown	NotBlank	TestTableRelation
DrillDownPageID	Numeric	ValidateTableRelation

Editable
ExtendedDatatype

PageType
PartType

ValuesAllowed

Most of these are typically not properties changed through customization as they can have a negative effect on the Microsoft Dynamics NAV deployment.

Restricted Properties on Existing Table Modifications

You cannot change the values for the following properties for existing tables and fields in an extension.

Table Properties	Field Properties	
Name	Name	NotBlank
DataPerCompany	AccessByPermission	Numeric
Permissions	Compressed	Owner
LookupPageID	Data Type	SQL Data Type
DrillDownPageID	DataLength	SubType (BLOB)
PastelsValid	DateFormula	TableIDExpr
LinkedObject	ExtendedDataType	TableRelation
TableType	FieldClass	TestTableRelation
	MaxValue	ValidateTableRelation
	MinValue	ValuesAllowed Width

You can add table keys, but you cannot delete or modify existing keys.

You can add fields to a table group, but you cannot remove fields or groups.

Extension example

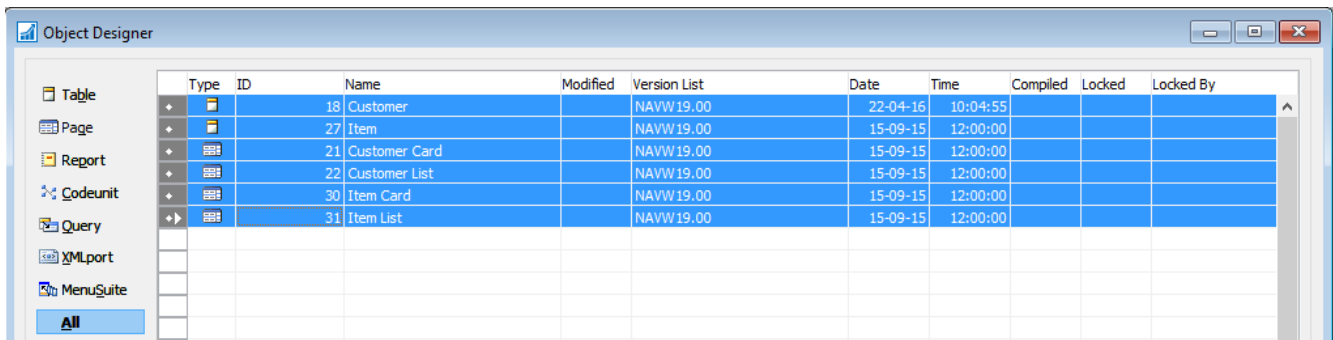
One example of an application that can function as an extension could be an add-on module to the sales module handling subscriptions. Subscriptions in Dynamics NAV always prove tricky to implement primarily because they are recurring but also because they cannot represent a inventory value if items are used for the invoicing.

The functional requirements are:

- 1) A new tabel: Subscriptions must be created to identify each type of subscription
The Table must hold the following information:
 - Subscription code
 - Subscription text
 - Item Number
 - Invoicing Schedule
 - Invoicing Frequence
 - Invoicing Price
 - Deferral Code
- 2) Card pages and list pages for the subscriptions
- 3) A Fact Box to show the number of active subscriptions etc.
- 4) A customer must be assigned one or mors subscriptions and therefore another table must be added:

- Customer No
 - Subscription Code
 - Item No
 - Created date
 - Last Invoice Date
 - Next Invoice Date
 - Cancelled Date
 - Active
 - Invoicing Price
 - Allow Line Discount
- 5) A List page for the customer subscriptions
 - 6) A Fact Box to show the active subscriptions etc.
 - 7) New Field on the Customer to show if the customer is an active subscription customer
This is to be able to make a Cue for subscription customers later
 - 8) The customer card and list must be altered with extra fields, actions and the FactBox.
 - 9) New Field on the Customer to show if the customer is an active subscription customer
This is to be able to make a Cue for subscription customers later
 - 10) The Item card and list must be altered with extra fields, actions and the FactBox.
 - 11) A codeunit to create the invoices
 - 12) Make a subscription event to update the subscription customer table when posted
 - 13) A New MenuSuite must be created with the subscription menu items

The first thing to do is to create a backup of all objects that will be affected by the changes:



Type	ID	Name	Modified	Version List	Date	Time	Compiled	Locked	Locked By
Table	18	Customer		NAVW 19.00	22-04-16	10:04:55			
Table	27	Item		NAVW 19.00	15-09-15	12:00:00			
Report	21	Customer Card		NAVW 19.00	15-09-15	12:00:00			
Report	22	Customer List		NAVW 19.00	15-09-15	12:00:00			
Codeunit	30	Item Card		NAVW 19.00	15-09-15	12:00:00			
Query	31	Item List		NAVW 19.00	15-09-15	12:00:00			

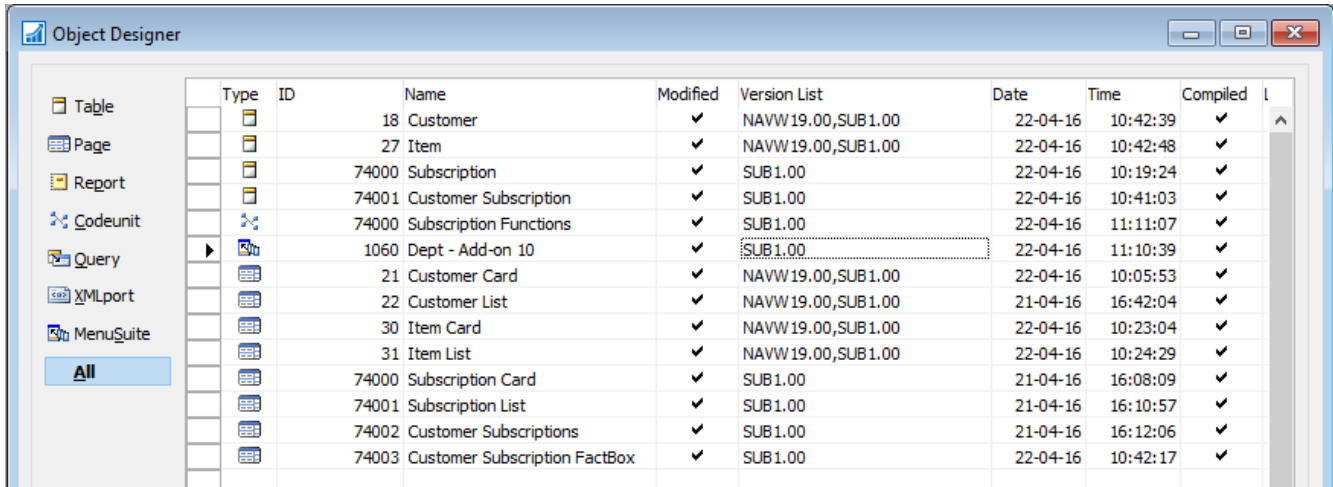
The objects must be exported in a text format to be used when building the delta files.

The original are stored in a newly created folder structure:

 Changed	22-04-2016 16:54
 Delta	22-04-2016 16:54
 Originals	22-04-2016 16:54

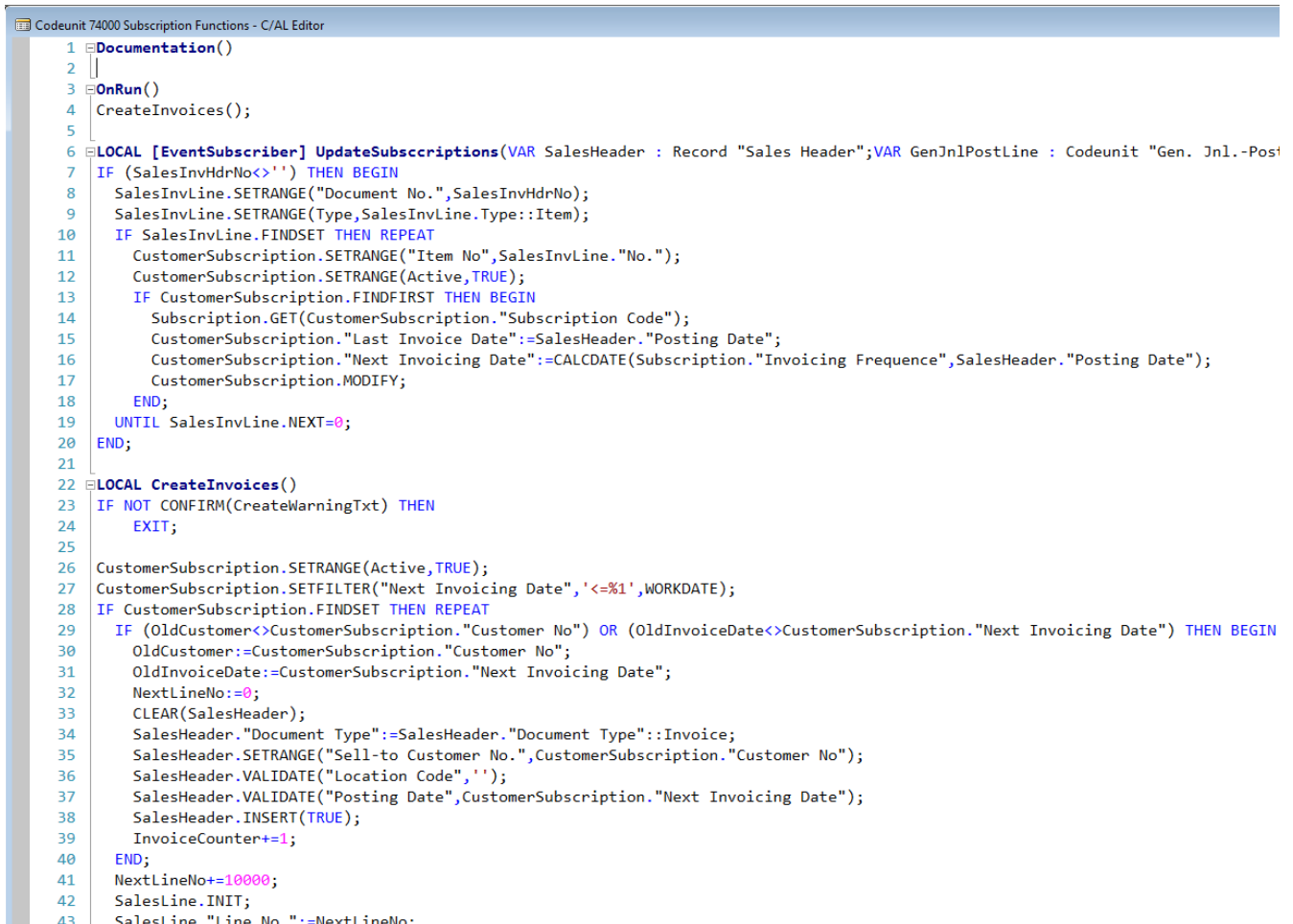
Development

Next the application is built:



Type	ID	Name	Modified	Version List	Date	Time	Compiled
	18	Customer	✓	NAVW 19.00,SUB1.00	22-04-16	10:42:39	✓
	27	Item	✓	NAVW 19.00,SUB1.00	22-04-16	10:42:48	✓
	74000	Subscription	✓	SUB1.00	22-04-16	10:19:24	✓
	74001	Customer Subscription	✓	SUB1.00	22-04-16	10:41:03	✓
	74000	Subscription Functions	✓	SUB1.00	22-04-16	11:11:07	✓
	1060	Dept - Add-on 10	✓	SUB1.00	22-04-16	11:10:39	✓
	21	Customer Card	✓	NAVW 19.00,SUB1.00	22-04-16	10:05:53	✓
	22	Customer List	✓	NAVW 19.00,SUB1.00	21-04-16	16:42:04	✓
	30	Item Card	✓	NAVW 19.00,SUB1.00	22-04-16	10:23:04	✓
	31	Item List	✓	NAVW 19.00,SUB1.00	22-04-16	10:24:29	✓
	74000	Subscription Card	✓	SUB1.00	21-04-16	16:08:09	✓
	74001	Subscription List	✓	SUB1.00	21-04-16	16:10:57	✓
	74002	Customer Subscriptions	✓	SUB1.00	21-04-16	16:12:06	✓
	74003	Customer Subscription FactBox	✓	SUB1.00	22-04-16	10:42:17	✓

All changes to the code has been added as functions and subscriptions events in a codeunit:



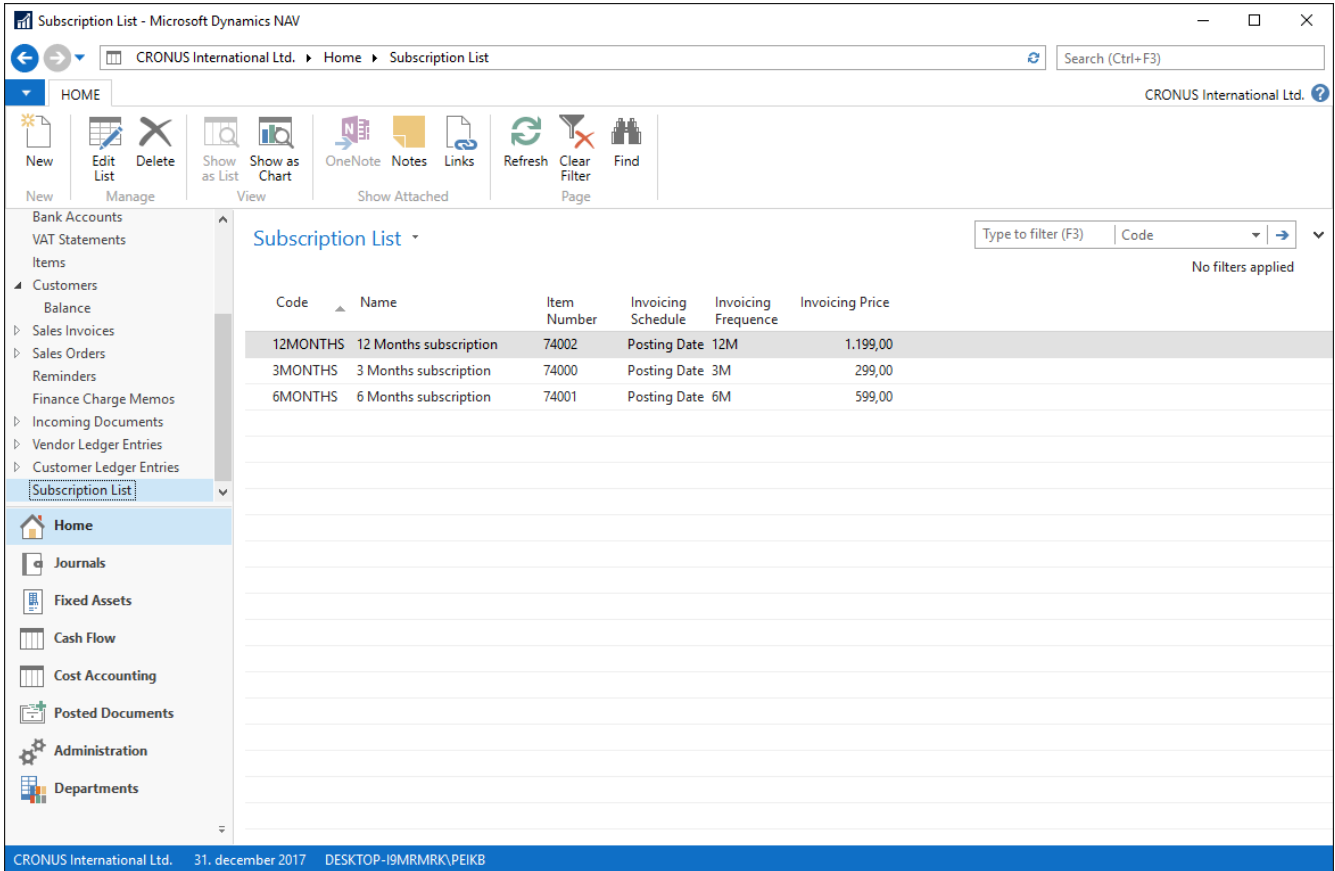
```

1 Documentation()
2 |
3 OnRun()
4 CreateInvoices();
5
6 LOCAL [EventSubscriber] UpdateSubscriptions(VAR SalesHeader : Record "Sales Header";VAR GenJnlPostLine : Codeunit "Gen. Jnl.-Post
7 IF (SalesInvHdrNo<>'') THEN BEGIN
8 SalesInvLine.SETRANGE("Document No.",SalesInvHdrNo);
9 SalesInvLine.SETRANGE(Type,SalesInvLine.Type::Item);
10 IF SalesInvLine.FINDSET THEN REPEAT
11 CustomerSubscription.SETRANGE("Item No",SalesInvLine."No.");
12 CustomerSubscription.SETRANGE(Active,TRUE);
13 IF CustomerSubscription.FINDFIRST THEN BEGIN
14 Subscription.GET(CustomerSubscription."Subscription Code");
15 CustomerSubscription."Last Invoice Date":=SalesHeader."Posting Date";
16 CustomerSubscription."Next Invoicing Date":=CALCDATE(Subscription."Invoicing Freqeunce",SalesHeader."Posting Date");
17 CustomerSubscription.MODIFY;
18 END;
19 UNTIL SalesInvLine.NEXT=0;
20 END;
21
22 LOCAL CreateInvoices()
23 IF NOT CONFIRM(CreateWarningTxt) THEN
24 EXIT;
25
26 CustomerSubscription.SETRANGE(Active,TRUE);
27 CustomerSubscription.SETFILTER("Next Invoicing Date",'<=%1',WORKDATE);
28 IF CustomerSubscription.FINDSET THEN REPEAT
29 IF (OldCustomer<>CustomerSubscription."Customer No") OR (OldInvoiceDate<>CustomerSubscription."Next Invoicing Date") THEN BEGIN
30 OldCustomer:=CustomerSubscription."Customer No";
31 OldInvoiceDate:=CustomerSubscription."Next Invoicing Date";
32 NextLineNo:=0;
33 CLEAR(SalesHeader);
34 SalesHeader."Document Type":=SalesHeader."Document Type"::Invoice;
35 SalesHeader.SETRANGE("Sell-to Customer No.",CustomerSubscription."Customer No");
36 SalesHeader.VALIDATE("Location Code",'');
37 SalesHeader.VALIDATE("Posting Date",CustomerSubscription."Next Invoicing Date");
38 SalesHeader.INSERT(TRUE);
39 InvoiceCounter+=1;
40 END;
41 NextLineNo+=10000;
42 SalesLine.INIT;
43 SalesLine."Line No.":=NextLineNo;

```

The testing

Three new subscriptions:



Subscription List - Microsoft Dynamics NAV

CRONUS International Ltd. Home Subscription List

Search (Ctrl+F3)

HOME CRONUS International Ltd.

New Edit List Delete Show as List Show as Chart OneNote Notes Links Refresh Clear Filter Find

Bank Accounts VAT Statements Items Customers Balance Sales Invoices Sales Orders Reminders Finance Charge Memos Incoming Documents Vendor Ledger Entries Customer Ledger Entries Subscription List

Home Journals Fixed Assets Cash Flow Cost Accounting Posted Documents Administration Departments

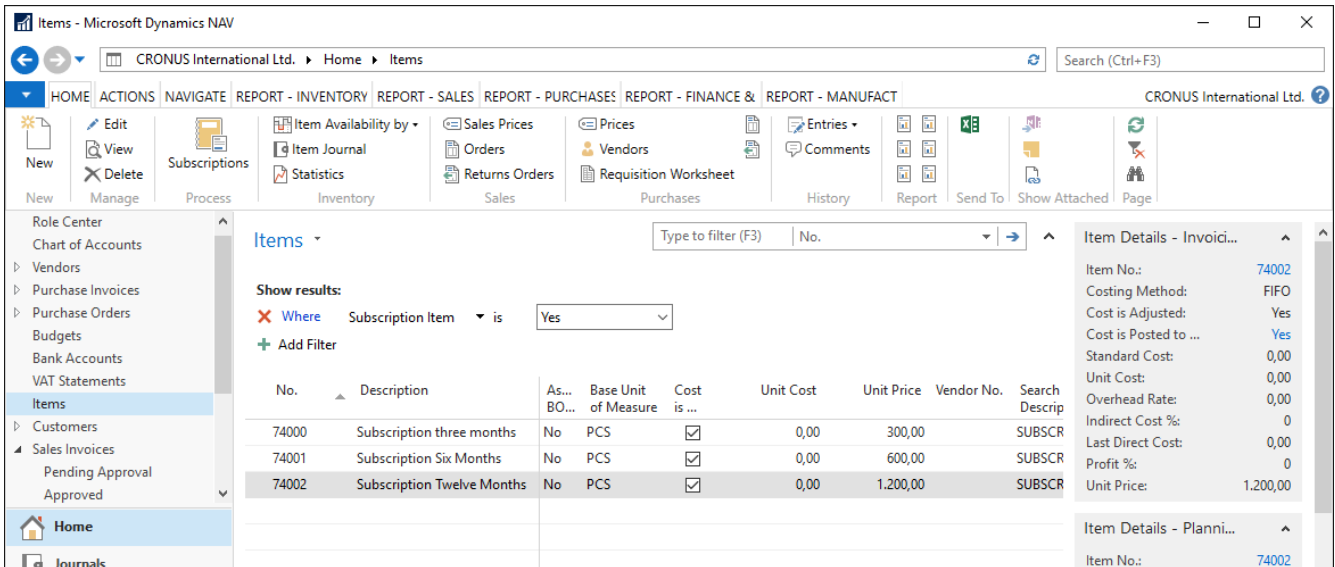
Subscription List

Type to filter (F3) Code No filters applied

Code	Name	Item Number	Invoicing Schedule	Invoicing Frequency	Invoicing Price
12MONTHS	12 Months subscription	74002	Posting Date	12M	1,199,00
3MONTHS	3 Months subscription	74000	Posting Date	3M	299,00
6MONTHS	6 Months subscription	74001	Posting Date	6M	599,00

CRONUS International Ltd. 31. december 2017 DESKTOP-19MRMRK\PEIKB

A number of items are created:



Items - Microsoft Dynamics NAV

CRONUS International Ltd. Home Items

Search (Ctrl+F3)

HOME ACTIONS NAVIGATE REPORT - INVENTORY REPORT - SALES REPORT - PURCHASES REPORT - FINANCE & REPORT - MANUFACT CRONUS International Ltd.

New Edit View Subscriptions Item Availability by Sales Prices Prices Entries Item Journal Orders Vendors Comments Item Journal Returns Orders Requisition Worksheet Statistics

Role Center Chart of Accounts Vendors Purchase Invoices Purchase Orders Budgets Bank Accounts VAT Statements Items Customers Sales Invoices Pending Approval Approved Journals

Items

Type to filter (F3) No.

Show results: Where Subscription Item is Yes

No.	Description	As... BO...	Base Unit of Measure	Cost is ...	Unit Cost	Unit Price	Vendor No.	Search Descrip
74000	Subscription three months	No	PCS	<input checked="" type="checkbox"/>	0,00	300,00		SUBSCR
74001	Subscription Six Months	No	PCS	<input checked="" type="checkbox"/>	0,00	600,00		SUBSCR
74002	Subscription Twelve Months	No	PCS	<input checked="" type="checkbox"/>	0,00	1,200,00		SUBSCR

Item Details - Invoic...
 Item No.: 74002
 Costing Method: FIFO
 Cost is Adjusted: Yes
 Cost is Posted to ... Yes
 Standard Cost: 0,00
 Unit Cost: 0,00
 Overhead Rate: 0,00
 Indirect Cost %: 0
 Last Direct Cost: 0,00
 Profit %: 0
 Unit Price: 1,200,00

Item Details - Planni...
 Item No.: 74002

Then a number of customers are assigned subscriptions:

The screenshot shows the 'Customers' list in Microsoft Dynamics NAV. A filter is applied: 'Where Subscription Customer is Yes'. The list contains three customers:

No.	Name	Respon... Center	Location Code	Phone No.	Contact	Search Name
10000	The Cannon Group PLC	BIRMINGH...	BLUE		Mr. Andy Teal	THE CA
20000	Selangorian Ltd.				Mr. Mark McArthur	SELANG
30000	John Haddock Insurance Co.				Miss Patricia Doyle	JOHN H

Each customer has been assigned at least one active subscription:

The screenshot shows the 'Edit - Customer Subscriptions' window for customer 10000. It displays a table of active subscriptions:

Custo... No	Subscription Code	Item No	Start Date	Last Invoice Date	Next Invoic...	Cancelled Date	Active	Invoicing Price
10000	12MONTHS	74002	01-01-2017		01-01-2017		<input checked="" type="checkbox"/>	1,199,00

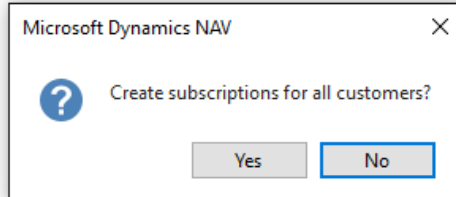
Now it is possible to create invoices:

- Departments
- ▶ Financial Management
- ▶ Sales & Marketing
- ▲ Subscriptions
 - Subscriptions
 - Periodic Activities
- ▶ Purchase
- ▶ Warehouse
- ▶ Manufacturing
- Jobs
- Resource Planning
- ▶ Service
- Human Resources
- ▶ Administration

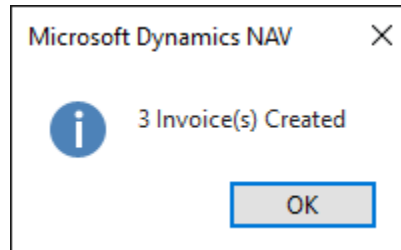
Periodic Activities

Tasks

Create Subscription Invoices



And the confirmation:



The Invoices:

Sales Invoices

Type to filter (F3) | No. | →

Limit totals: '..30-12-17

No.	Sell-to Custom...	Sell-to Customer Name	External Docume...	Location Code	Assigned User ID
1020	10000	The Cannon Group PLC		BLUE	
1021	20000	Selangorian Ltd.			
1022	30000	John Haddock Insurance Co.			

The invoice looks like a normal invoice:

1020 · The Cannon Group PLC

General ⚙️ ^

Sell-to Customer No.: <input type="text" value="10000"/>	Posting Date: <input type="text" value="01-01-2017"/>
Sell-to Contact No.: <input type="text" value="CT000007"/>	Document Date: <input type="text" value="01-01-2017"/>
Sell-to Customer Name: <input type="text" value="The Cannon Group PLC"/>	External Document No.: <input type="text"/>
Sell-to City: <input type="text" value="Birmingham"/>	Salesperson Code: <input type="text" value="PS"/>
Sell-to Contact: <input type="text" value="Mr. Andy Teal"/>	Status: <input type="text" value="Open"/>
Your Reference: <input type="text"/>	

^ Show more fields

Lines ^

⚡ Functions 📄 Line 📄 New 🔍 Find 🔍 Filter 🗑️ Clear Filter

Type	No.	Description	Location Code	Quantity	Unit of Measur...	Unit Price Excl. VAT	Li
Item	74000	Subscription three months	BLUE	1	PCS	300,00	

Invoice Discount Amount: <input type="text" value="0,00"/>	Total Excl. VAT (GBP): <input type="text" value="300,00"/>
Invoice Discount %: <input type="text" value="0"/>	Total VAT (GBP): <input type="text" value="0,00"/>
	Total Incl. VAT (GBP): <input type="text" value="300,00"/>

But posting it will update the CustomerSubscription table:

Edit - Customer Subscriptions — □ ×

HOME CRONUS International Ltd. ?

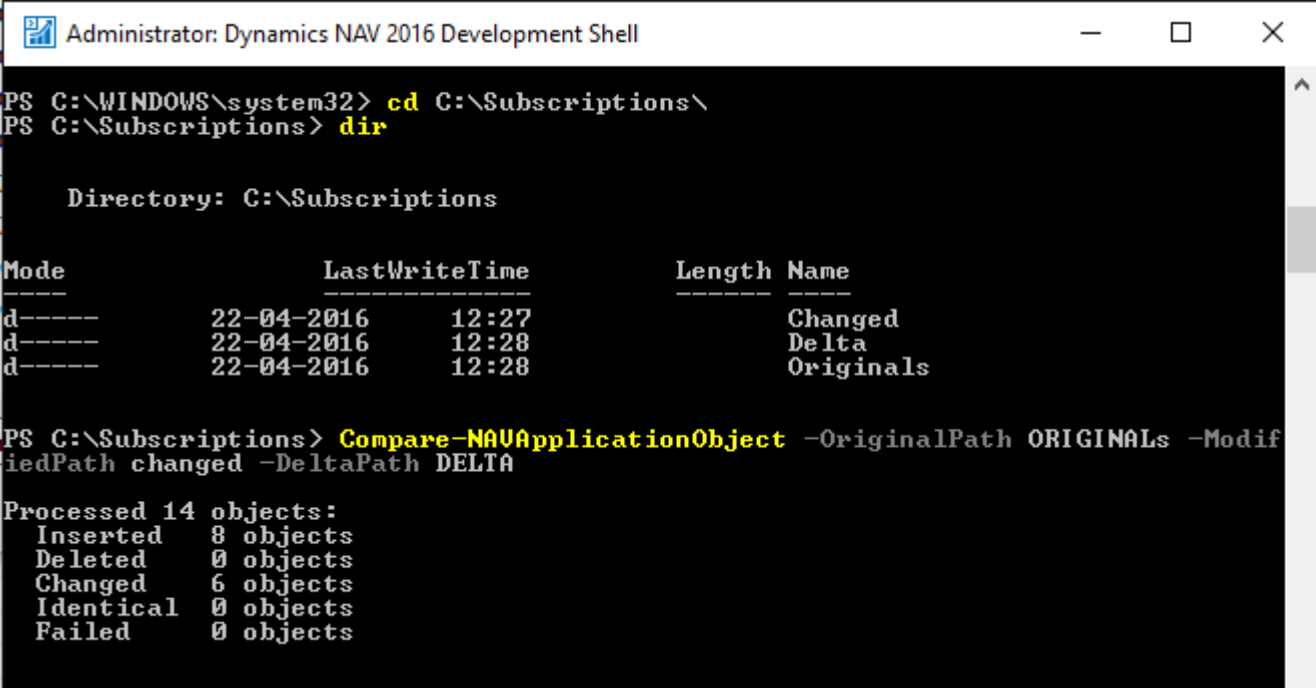
Custo... No	Subscription Code	Item No	Start Date	Last Invoice Date	Next Invoicin...	Cancelled Date	Active	Invoicing Price
10000	12MONTHS	74002	01-01-2017	01-01-2017	01-01-2018		<input checked="" type="checkbox"/>	1.199,00

OK

Mission Accomplished.

Creating the navx package

The first thing is to create the delta files:



```
Administrator: Dynamics NAV 2016 Development Shell
PS C:\WINDOWS\system32> cd C:\Subscriptions\
PS C:\Subscriptions> dir

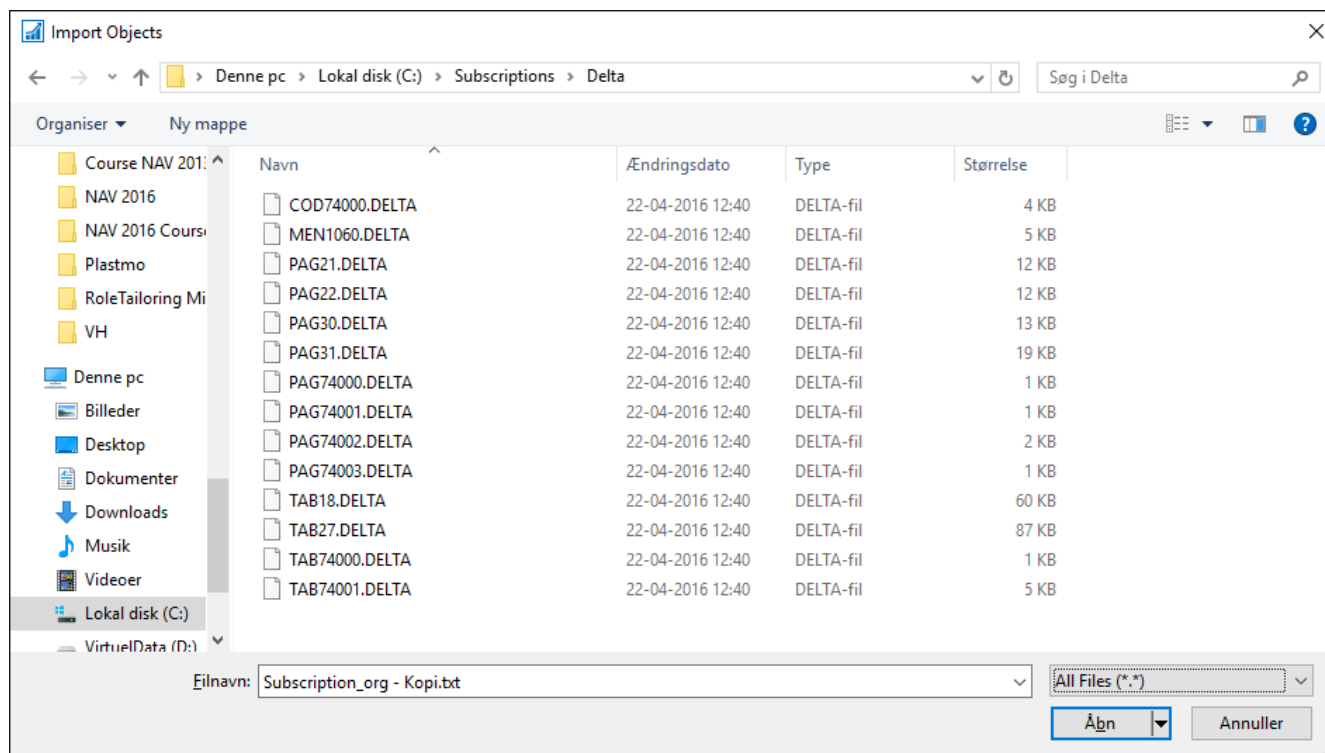
Directory: C:\Subscriptions

Mode                LastWriteTime         Length Name
----                -
d-----          22-04-2016   12:27         Changed
d-----          22-04-2016   12:28         Delta
d-----          22-04-2016   12:28         Originals

PS C:\Subscriptions> Compare-NAVApplicationObject -OriginalPath ORIGINALs -ModifiedPath changed -DeltaPath DELTA

Processed 14 objects:
  Inserted      8 objects
  Deleted       0 objects
  Changed       6 objects
  Identical     0 objects
  Failed        0 objects
```

This will create a number of files:



Then the manifest must be created:

```
Administrator: Dynamics NAV 2016 Development Shell
PS C:\Subscriptions> New-NAVAppManifest -Name "Subscription" -Publisher "CRONUS Development, Inc." -Version "1.0.0.0" | New-NAVAppManifestFile -Path '.\Manifest-Subscription.xml' -Force
PS C:\Subscriptions>
```

In this case the manifest is created and then "piped" into the next command, which will save the manifest as a file.

The file looks like this:

```
C:\Subscriptions\Manifest-Sub
<?xml version="1.0"?>
- <Package xmlns="http://schemas.microsoft.com/navx/2015/manifest">
  <App CompatibilityId="1.0.0.0" Version="1.0.0.0" Description="" Publisher="CRONUS Development, Inc." Name="Subscription" Id="eddd0109-cbf6-48f4-ba77-f229c6ba2251"/>
  <Capabilities/>
- <Prerequisites>
  <Objects/>
  </Prerequisites>
  <Dependencies/>
</Package>
```

Lastly the package must be created:

```
Administrator: Dynamics NAV 2016 Development Shell
PS C:\Subscriptions> Get-NAVAppManifest -Path '.\Subscription.xml' | New-NAVAppPackage -Path '.\Subscription.navx' -SourcePath '.\Delta\ -Force
PS C:\Subscriptions>
```

And the navx package is ready:

Navn	Ændringsdato	Type	Størrelse
Changed	22-04-2016 16:54	Filmappe	
Delta	22-04-2016 16:54	Filmappe	
Originals	22-04-2016 16:54	Filmappe	
Manifest-Subscription.xml	22-04-2016 13:42	XML-fil	1 KB
Subscription.navx	22-04-2016 17:00	NAVX-fil	76 KB



Publishing and installing the package

The next thing to do is to publish the package to the server instance where it is needed. In this case the server instance is: DynamicsNAV90 which is a totally clean database.

```
Administrator: Dynamics NAV 2016 Development Shell
PS C:\WINDOWS\system32> cd C:\Subscriptions\
PS C:\Subscriptions> Publish-NAVApp -ServerInstance DynamicsNAV90 -Path Subscription.navx
PS C:\Subscriptions>
```

To see, which apps are published on the server instance, use the GetNAVAppInfo Cmdlet:

```
Administrator: Dynamics NAV 2016 Development Shell
PS C:\Subscriptions> Get-NAVAppInfo -ServerInstance DynamicsNAV90

Id                : de865612-9f14-4140-a9e0-06f2f2a6bbe0
Name              : Subscription
Version          : 1.0.0.0
Publisher        : CRONUS Development, Inc.
Description      :
Compatibility Id : 1.0.0.0
Capabilities     : UiChanges
                  UiAdds
                  SchemaChanges
                  SchemaAdds
                  CodeAdds

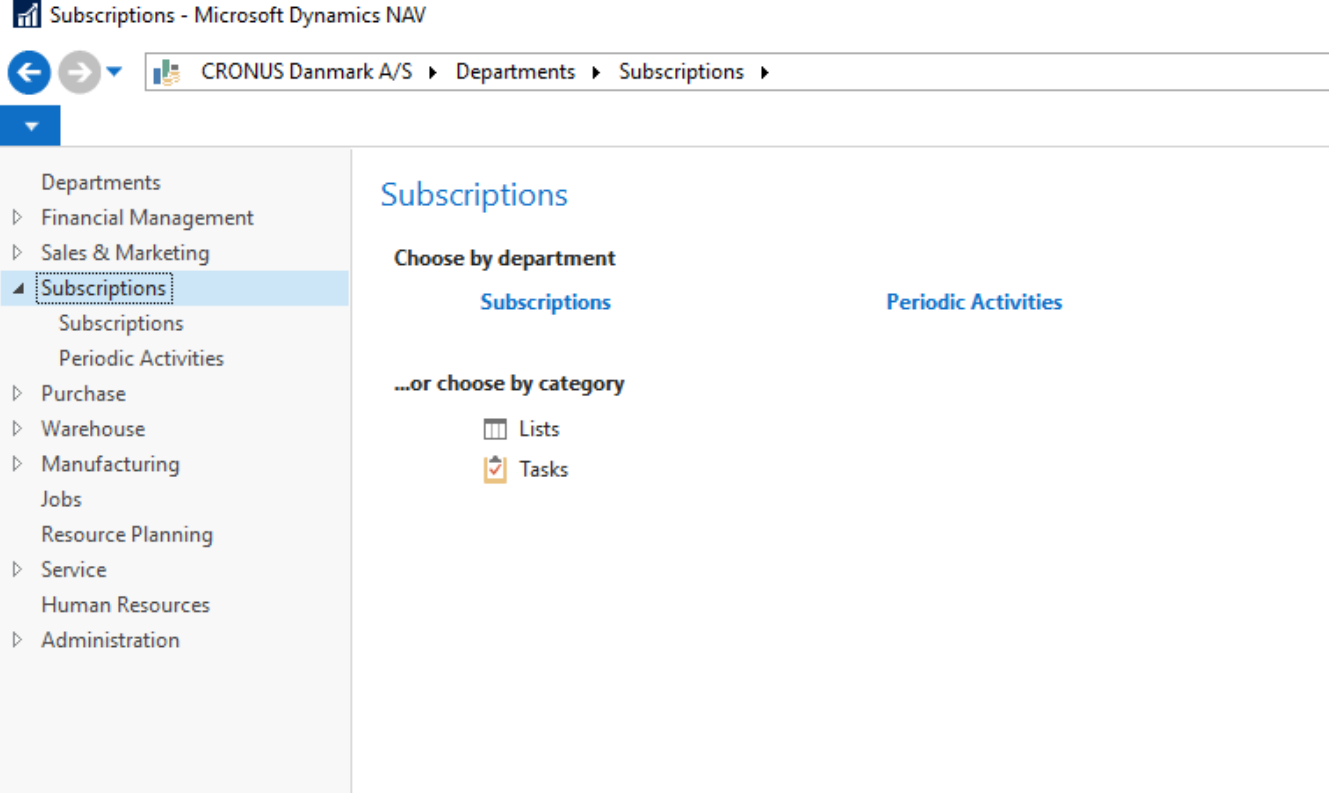
Prerequisites    :
Dependencies     :
```

Lastly the package must be installed

```
Administrator: Dynamics NAV 2016 Development Shell
PS C:\Subscriptions> Install-NAVApp -ServerInstance DynamicsNAV90 -Name "Subscription" -Tenant Default
PS C:\Subscriptions>
```

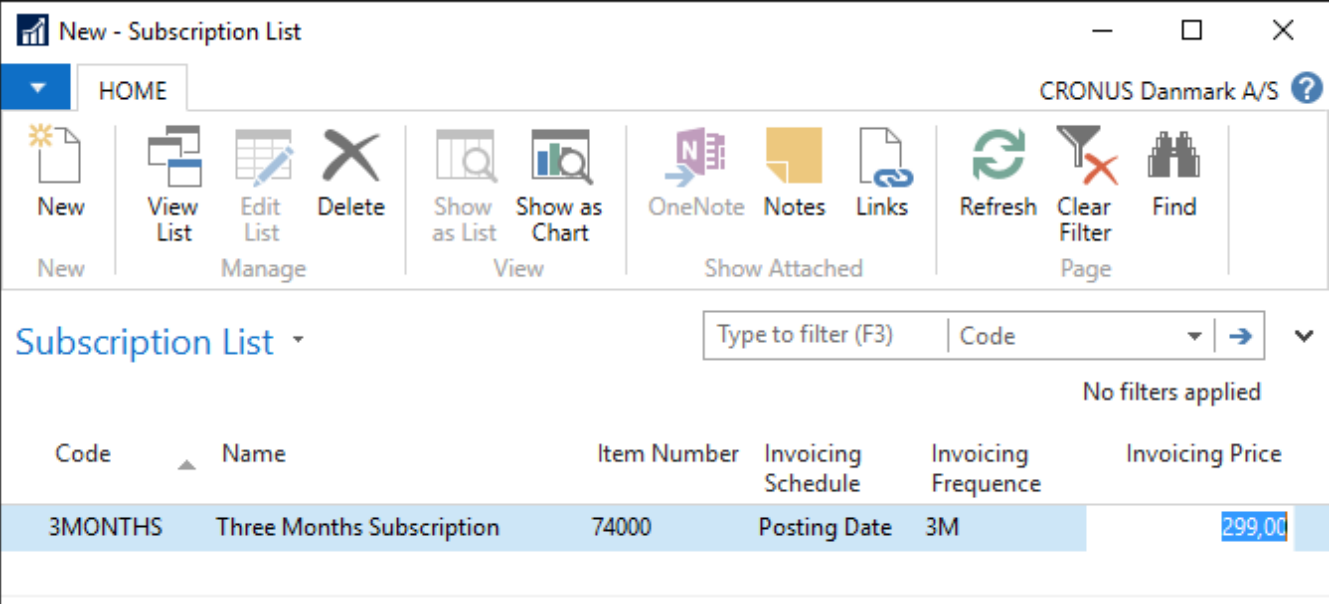
Testing the App

Opening the DynamicsNAV90, the Departments menu is the place to look:



The screenshot shows the Dynamics NAV interface. At the top, the breadcrumb path is "CRONUS Danmark A/S > Departments > Subscriptions". The left-hand navigation pane is open to the "Subscriptions" menu item, which is highlighted. Below it, sub-items include "Subscriptions", "Periodic Activities", "Purchase", "Warehouse", "Manufacturing", "Jobs", "Resource Planning", "Service", "Human Resources", and "Administration". The main content area displays the "Subscriptions" page with options to "Choose by department" (Subscriptions, Periodic Activities) and "...or choose by category" (Lists, Tasks).

It is possible to create subscriptions:



The screenshot shows the "New - Subscription List" window. The ribbon includes "HOME" and various actions: "New", "View List", "Edit List", "Delete", "Show as List", "Show as Chart", "OneNote", "Notes", "Links", "Refresh", "Clear Filter", and "Find". Below the ribbon, there is a search bar with "Type to filter (F3)" and a dropdown menu for "Code". The table below shows one subscription entry:

Code	Name	Item Number	Invoicing Schedule	Invoicing Frequency	Invoicing Price
3MONTHS	Three Months Subscription	74000	Posting Date	3M	299,00

Building workflows

What is a workflow?

Create a workflow event

Create a workflow response

Enable the workflow & response in the NAV system

This chapter is a compilation of available Microsoft material. More information can be found at

[https://msdn.microsoft.com/en-us/library/mt574349\(v=nav.90\).aspx](https://msdn.microsoft.com/en-us/library/mt574349(v=nav.90).aspx)



What are workflows?

- A workflow is a movement of steps or tasks through a work process
- Workflows help connect business processes to best practices or industry-standard practices
- For example, ensuring a customer's credit limit has been verified
- For example, requiring a two-person approval of a significant vendor payment
- In Microsoft Dynamics NAV 2016, a workflow is likely to contain steps related to approval, notification, or business-process automation





Workflows

Benefits

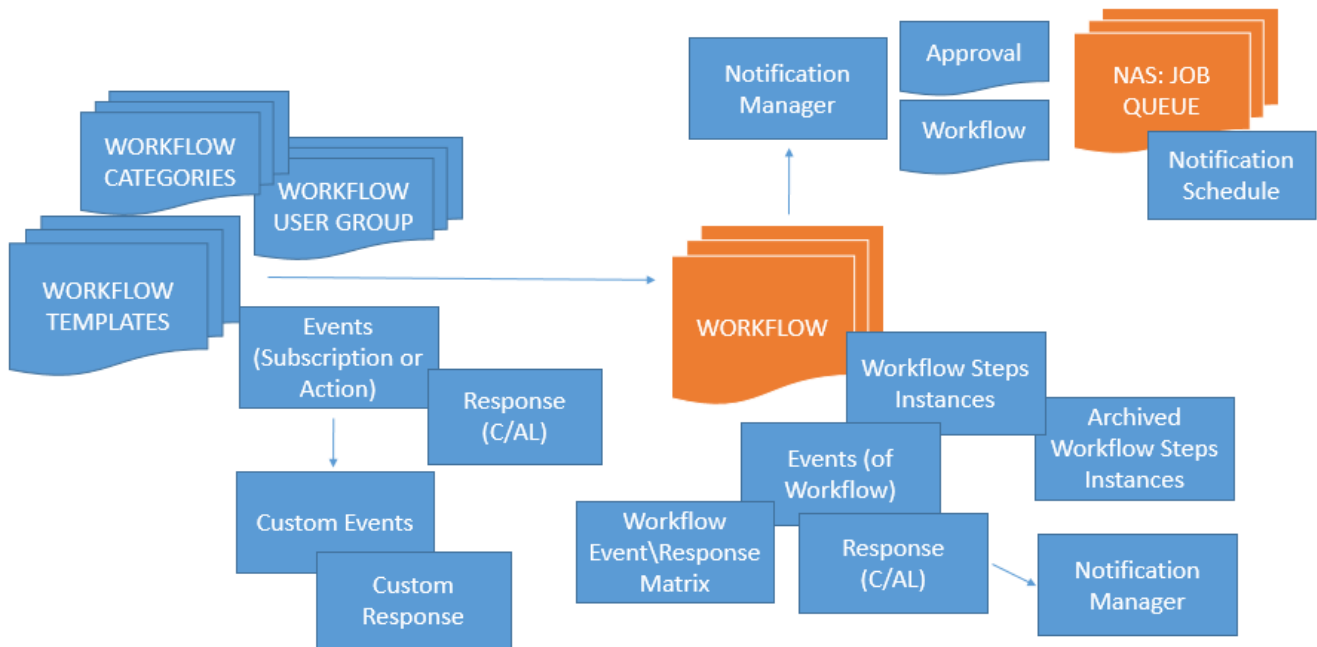
Out-of-the-box workflow templates, with situation-specific parameters, are ready in just a few clicks, facilitating sales & purchase approvals and incoming document handling

Redesigned approvals functionality offers more capabilities and is incorporated in the workflow system to allow for easy extension

The workflow designer contains intuitive layout and terminology, such as “When Event” and “Then Response”, making it easy to model company processes

Email notifications and approvals by phone app mean that workflow users can be anyone in the organization and don’t require access to a PC to complete simple steps

Nav 2016 Workflow & Notification Schema



Categories: A grouping of the workflow template, e.g. Administration, Finance, Integration.

User Group: A group of approvers to approve the workflow either simultaneously or sequentially.

Template: Base for the actual workflow.

Event: The situation that triggers the workflow on a given condition.

Response: The action taken depending on the state.

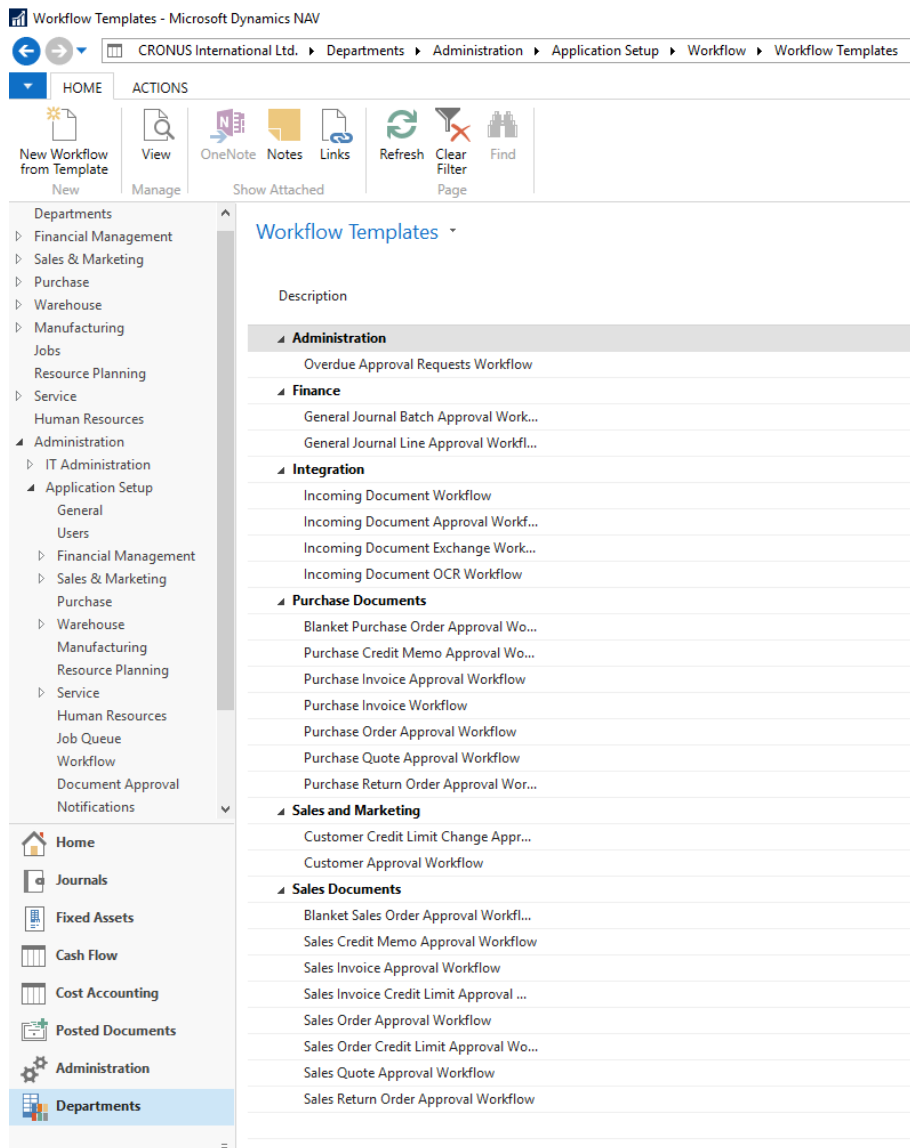
Workflow: The actual workflow that will be executed.

Notification: The message sent by email.

Approval: The approval request entries that requires an action from the approver.

Out-of-the-box workflows in Dynamics NAV 2016

- A workflow is a movement of steps or tasks through a work process
- Workflows help connect business processes to best practices or industry-standard practices
- For example, ensuring a customer's credit limit has been verified
- For example, requiring a two-person approval of a significant vendor payment
- In Microsoft Dynamics NAV 2016, a workflow is likely to contain steps related to approval, notification, or business-process automation

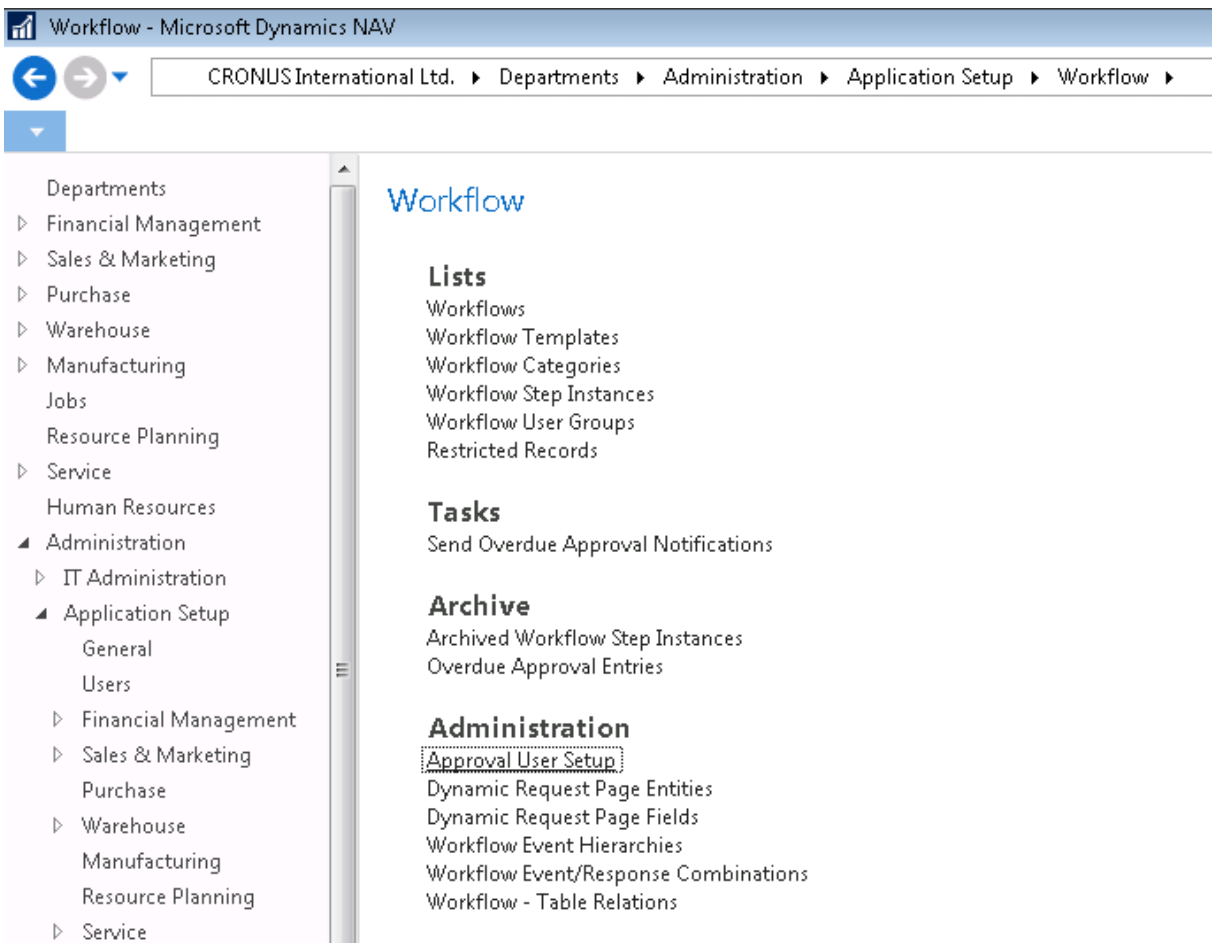


The screenshot shows the 'Workflow Templates' page in Microsoft Dynamics NAV 2016. The breadcrumb trail is: CRONUS International Ltd. > Departments > Administration > Application Setup > Workflow > Workflow Templates. The page features a ribbon with 'HOME' and 'ACTIONS' tabs. The 'ACTIONS' tab includes buttons for 'New Workflow from Template', 'View', 'OneNote', 'Notes', 'Links', 'Refresh', 'Clear Filter', and 'Find'. The left-hand navigation pane shows a tree view of departments, with 'Administration' expanded to show 'Application Setup' and 'Workflow'. The main content area displays a list of workflow templates, categorized by department:

Department	Workflow Name
Administration	Overdue Approval Requests Workflow
Finance	General Journal Batch Approval Work...
Finance	General Journal Line Approval Workf...
Integration	Incoming Document Workflow
Integration	Incoming Document Approval Workf...
Integration	Incoming Document Exchange Work...
Integration	Incoming Document OCR Workflow
Purchase Documents	Blanket Purchase Order Approval Wo...
Purchase Documents	Purchase Credit Memo Approval Wo...
Purchase Documents	Purchase Invoice Approval Workflow
Purchase Documents	Purchase Invoice Workflow
Purchase Documents	Purchase Order Approval Workflow
Purchase Documents	Purchase Quote Approval Workflow
Purchase Documents	Purchase Return Order Approval Wor...
Sales and Marketing	Customer Credit Limit Change Appr...
Sales and Marketing	Customer Approval Workflow
Sales Documents	Blanket Sales Order Approval Workf...
Sales Documents	Sales Credit Memo Approval Workflow
Sales Documents	Sales Invoice Approval Workflow
Sales Documents	Sales Invoice Credit Limit Approval ...
Sales Documents	Sales Order Approval Workflow
Sales Documents	Sales Order Credit Limit Approval Wo...
Sales Documents	Sales Quote Approval Workflow
Sales Documents	Sales Return Order Approval Workflow

Setting up for credit limit approval

- Set up users in approval hierarchy
- Create a workflow user group and apply the users to the group
- Create a new workflow from one of the existing templates
- Link the workflow to the workflow user group
- Test the workflow

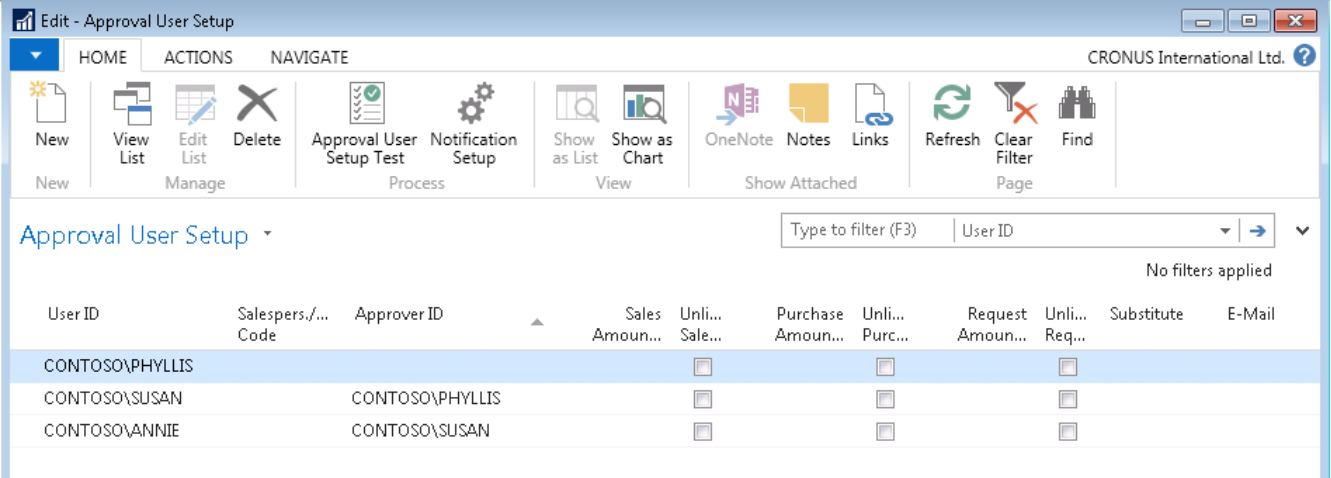


The screenshot shows the Microsoft Dynamics NAV interface for setting up a workflow. The breadcrumb navigation at the top reads: CRONUSInternational Ltd. > Departments > Administration > Application Setup > Workflow >. The left-hand navigation pane is expanded to 'Application Setup' > 'Users'. The main content area is titled 'Workflow' and is organized into three sections:

- Lists**
 - Workflows
 - Workflow Templates
 - Workflow Categories
 - Workflow Step Instances
 - Workflow User Groups
 - Restricted Records
- Tasks**
 - Send Overdue Approval Notifications
- Archive**
 - Archived Workflow Step Instances
 - Overdue Approval Entries
- Administration**
 - Approval User Setup
 - Dynamic Request Page Entities
 - Dynamic Request Page Fields
 - Workflow Event Hierarchies
 - Workflow Event/Response Combinations
 - Workflow - Table Relations

Setting up for credit limit approval

- Set up users in approval hierarchy

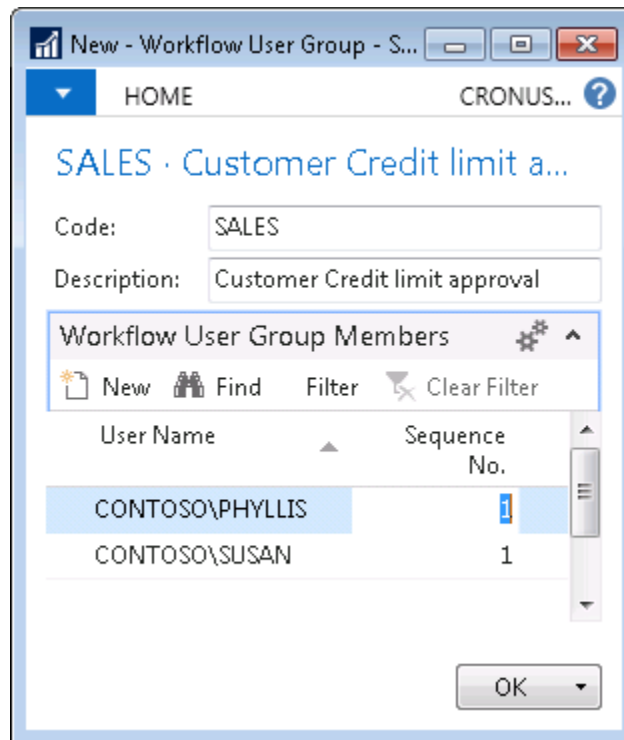


The screenshot shows the 'Edit - Approval User Setup' application window. The interface includes a ribbon with tabs for HOME, ACTIONS, and NAVIGATE. The ACTIONS tab is active, showing various icons for New, View List, Edit List, Delete, Approval User Setup Test, Notification Setup, Show as List, Show as Chart, OneNote, Notes, Links, Refresh, Clear Filter, and Find. Below the ribbon, there is a search bar with the text 'Type to filter (F3)' and a dropdown menu set to 'User ID'. The main area displays a table with the following data:

User ID	Salespers./... Code	Approver ID	Sales Amoun...	Unli... Sale...	Purchase Amoun...	Unli... Purc...	Request Amoun...	Unli... Req...	Substitute	E-Mail
CONTOSO\PHYLLIS				<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		
CONTOSO\SUSAN		CONTOSO\PHYLLIS		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		
CONTOSO\ANNIE		CONTOSO\SUSAN		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		

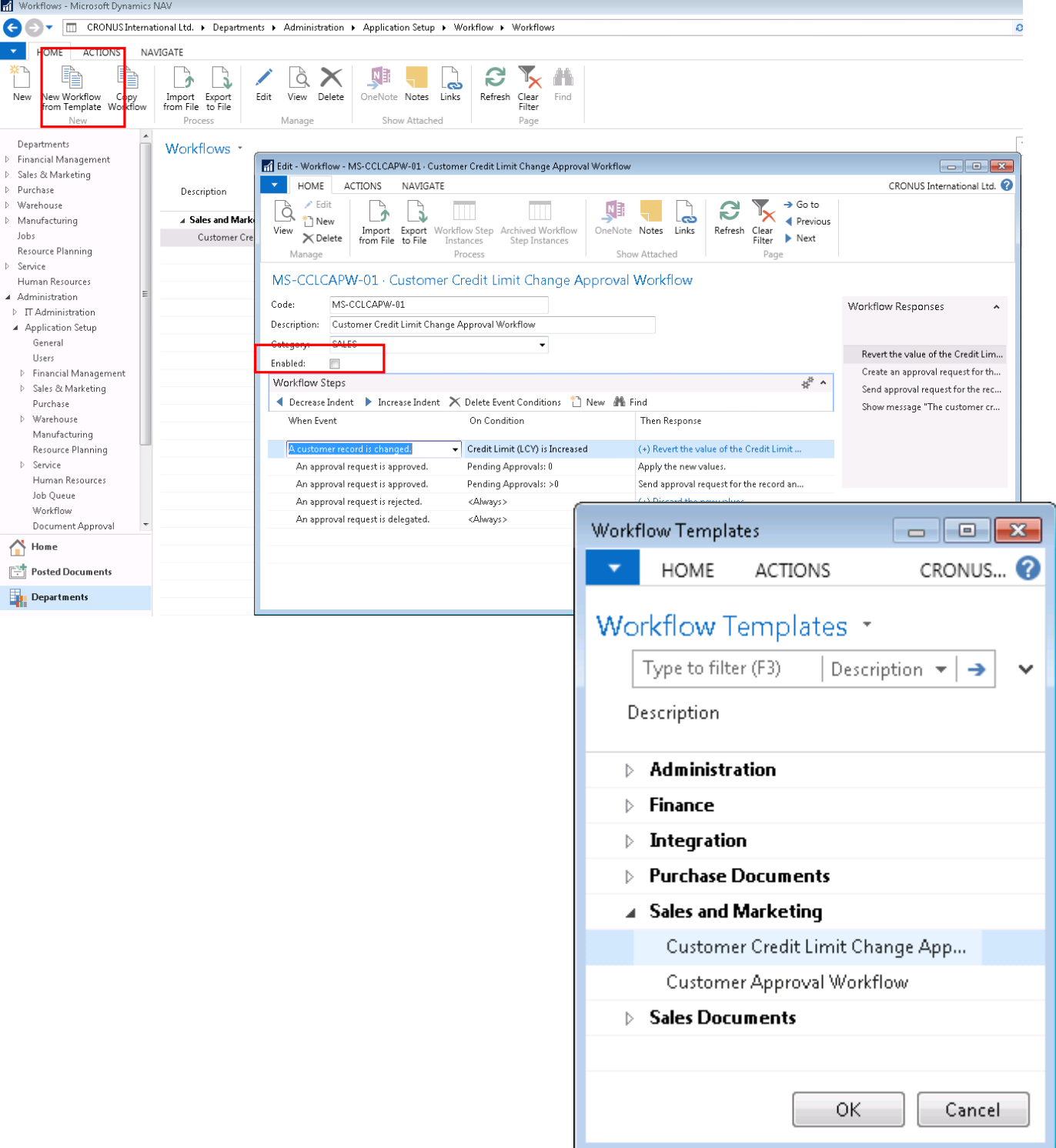
Setting up for credit limit approval

- Create a workflow user group and apply the users to the group
- Using the same Sequence No. the approval request will be sent to both approvers at the same time
- Using a higher Sequence No. for Phil, the approval request will first be sent for approval to Susan and later to Phil



Create a new workflow from one of the existing templates

- Using the “Create Workflow from template”



The screenshot displays the Microsoft Dynamics NAV Workflows interface. The main window is titled "Edit - Workflow - MS-CCLCAPW-01 - Customer Credit Limit Change Approval Workflow". The "ACTIONS" tab is active, and the "New Workflow from Template" button is highlighted with a red box. The "Enabled" checkbox is also highlighted with a red box. The "Workflow Steps" table is visible, showing a step with the event "A customer record is changed" and the condition "Credit Limit (LCY) is Increased".

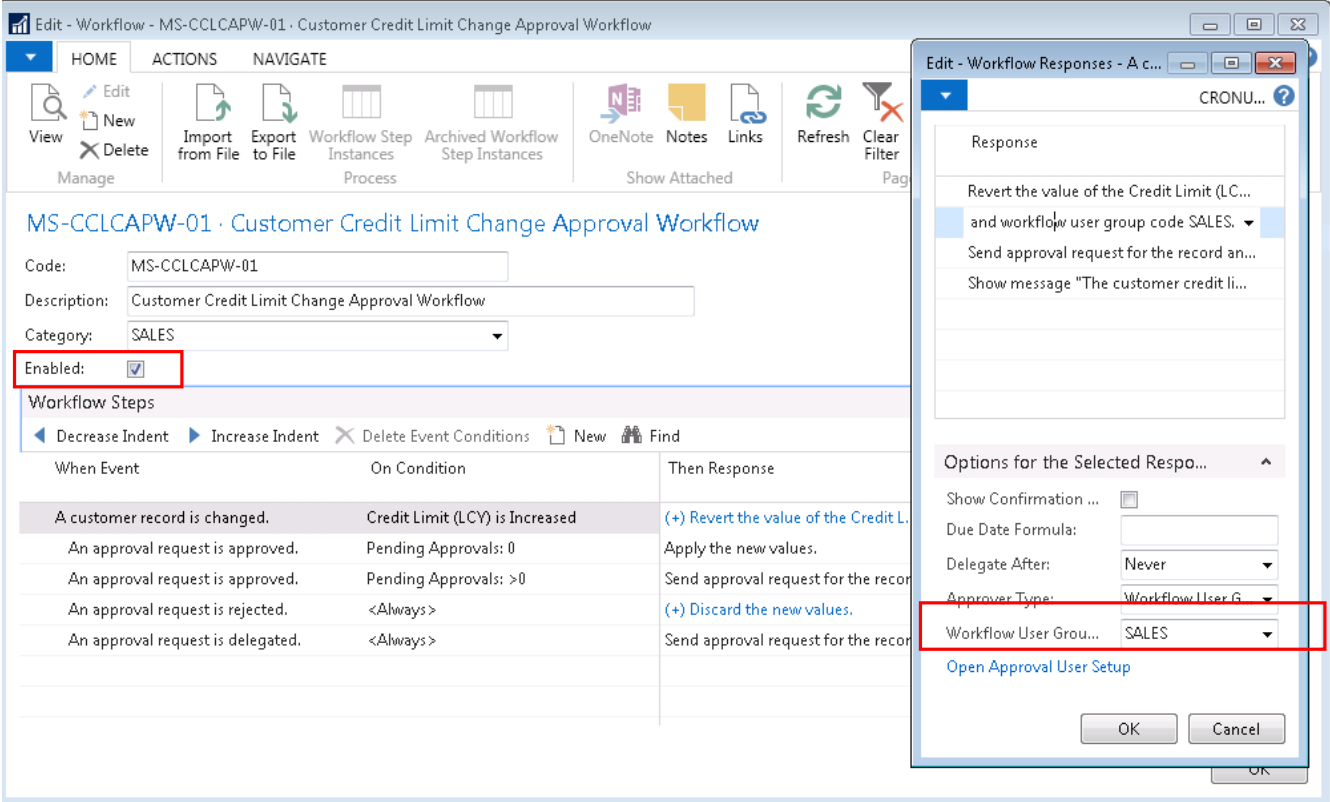
The "Workflow Templates" dialog box is open, showing a list of templates. The "Sales and Marketing" category is expanded, and the "Customer Credit Limit Change App..." template is selected. The dialog box includes a search bar, a list of templates, and "OK" and "Cancel" buttons.

When Event	On Condition	Then Response
A customer record is changed.	Credit Limit (LCY) is Increased	(+) Revert the value of the Credit Limit ...
An approval request is approved.	Pending Approvals: 0	Apply the new values.
An approval request is approved.	Pending Approvals: >0	Send approval request for the record an...
An approval request is rejected.	<Always>	(-) Discard the approval...
An approval request is delegated.	<Always>	

Category	Description
Administration	
Finance	
Integration	
Purchase Documents	
Sales and Marketing	
Customer Credit Limit Change App...	
Customer Approval Workflow	
Sales Documents	

Adjusting the workflow

- In the workflow, the approval group must be assigned to the workflow
- Then the workflow can be enabled



The screenshot displays the 'Edit - Workflow - MS-CCLCAPW-01 - Customer Credit Limit Change Approval Workflow' window. The 'Enabled' checkbox is checked and highlighted with a red box. The 'Workflow Steps' table is visible, and the 'Workflow Responses' dialog box is open, showing a response configuration for 'SALES'.

Workflow Steps Table:

When Event	On Condition	Then Response
A customer record is changed.	Credit Limit (LCY) is Increased	(+) Revert the value of the Credit L...
An approval request is approved.	Pending Approvals: 0	Apply the new values.
An approval request is approved.	Pending Approvals: >0	Send approval request for the recor...
An approval request is rejected.	<Always>	(+) Discard the new values.
An approval request is delegated.	<Always>	Send approval request for the recor...

Workflow Responses Dialog Box:

Response

Revert the value of the Credit Limit (LC...
and workflow user group code SALES. ▾
Send approval request for the record an...
Show message "The customer credit li...

Options for the Selected Respo... ^

Show Confirmation ...

Due Date Formula:

Delegate After: Never ▾

Approver Type: Workflow User G... ▾

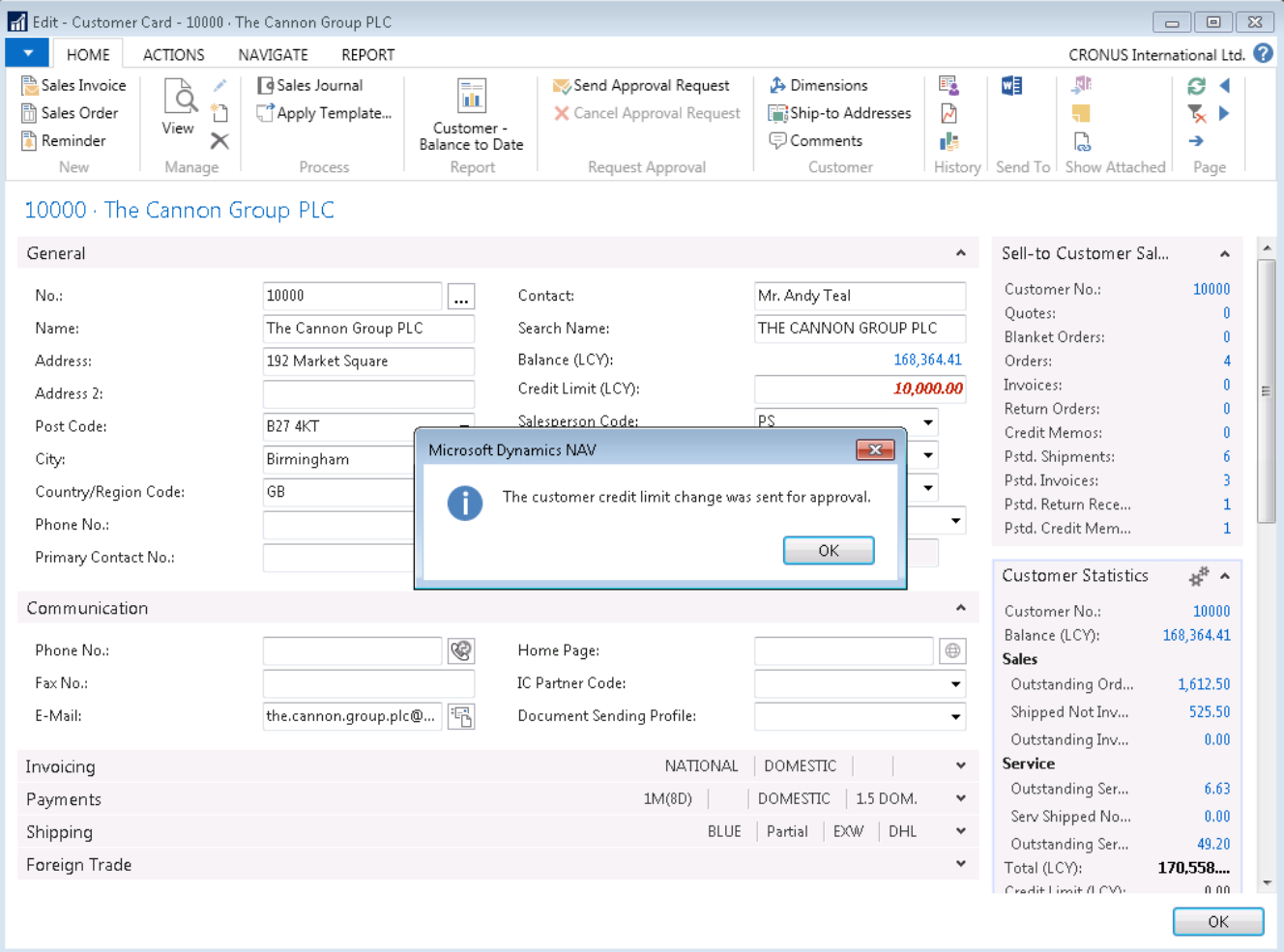
Workflow User Grou... SALES ▾

[Open Approval User Setup](#)

OK Cancel

Testing the workflow

- Logging on as Annie
- Annie changed the credit limit field to 10.000
- When she pressed ok the message appears



The screenshot shows the Microsoft Dynamics NAV interface for editing a customer card. The window title is "Edit - Customer Card - 10000 - The Cannon Group PLC". The ribbon includes tabs for HOME, ACTIONS, NAVIGATE, and REPORT. The ACTIONS tab is active, showing options like "Send Approval Request" and "Cancel Approval Request".

The main area displays the "General" section for customer "10000 - The Cannon Group PLC". Key fields include:

- No.: 10000
- Name: The Cannon Group PLC
- Address: 192 Market Square
- Post Code: B27 4KT
- City: Birmingham
- Country/Region Code: GB
- Contact: Mr. Andy Teal
- Search Name: THE CANNON GROUP PLC
- Balance (LCY): 168,364.41
- Credit Limit (LCY): 10,000.00
- Salesperson Code: PS

A modal dialog box titled "Microsoft Dynamics NAV" is overlaid on the screen, displaying the message: "The customer credit limit change was sent for approval." with an "OK" button.

On the right side, there are two summary panels:

- Sell-to Customer Sal...:** Shows various metrics such as Customer No. (10000), Quotes (0), Blanket Orders (0), Orders (4), Invoices (0), Return Orders (0), Credit Memos (0), Pstd. Shipments (6), Pstd. Invoices (3), Pstd. Return Rece... (1), and Pstd. Credit Mem... (1).
- Customer Statistics:** Shows Customer No. (10000), Balance (LCY) (168,364.41), Sales (Outstanding Ord... 1,612.50, Shipped Not Inv... 525.50, Outstanding Inv... 0.00), Service (Outstanding Ser... 6.63, Serv Shipped No... 0.00, Outstanding Ser... 49.20), Total (LCY) (170,558...), and Credit Limit (LCY) (0.00).

Testing the workflow

- Looking at the customer now, it is apparent that the credit limit has been reset pending the approval

10000 · The Cannon Group PLC

HOME ACTIONS NAVIGATE REPORT CRONUS International Ltd. ?

Sales Invoice Sales Order Reminder New
 View Manage
 Sales Journal Apply Template... Process
 Customer - Balance to Date Report
 Send Approval Request Cancel Approval Request Request Approval
 Dimensions Ship-to Addresses Comments Customer
 History Send To Show Attached Page

10000 · The Cannon Group PLC

General

No.: 10000
 Name: The Cannon Group PLC
 Address: 192 Market Square
 Address 2:
 Post Code: B27 4KT
 City: Birmingham
 Country/Region Code: GB
 Phone No.:
 Primary Contact No.:

Contact: Mr. Andy Teal
 Search Name: THE CANNON GROUP PLC
 Balance (LCY): 168,364.41
 Credit Limit (LCY): 0.00
 Salesperson Code: PS
 Responsibility Center: BIRMINGHAM
 Service Zone Code: M
 Blocked:
 Last Date Modified: 3/5/2016

Sell-to Customer Sal...

Customer No.: 10000
 Quotes: 0
 Blanket Orders: 0
 Orders: 4
 Invoices: 0
 Return Orders: 0
 Credit Memos: 0
 Pstd. Shipments: 6
 Pstd. Invoices: 3
 Pstd. Return Rece...: 1
 Pstd. Credit Mem...: 1

Communication

Phone No.:
 Fax No.:
 E-Mail: the.cannon.group.plc@cronusc...

Home Page:
 IC Partner Code:
 Document Sending Profile:

Invoicing NATIONAL | DOMESTIC |
Payments 1M(8D) | | DOMESTIC | 1.5 DOM.
Shipping BLUE | Partial | EXW | DHL
Foreign Trade

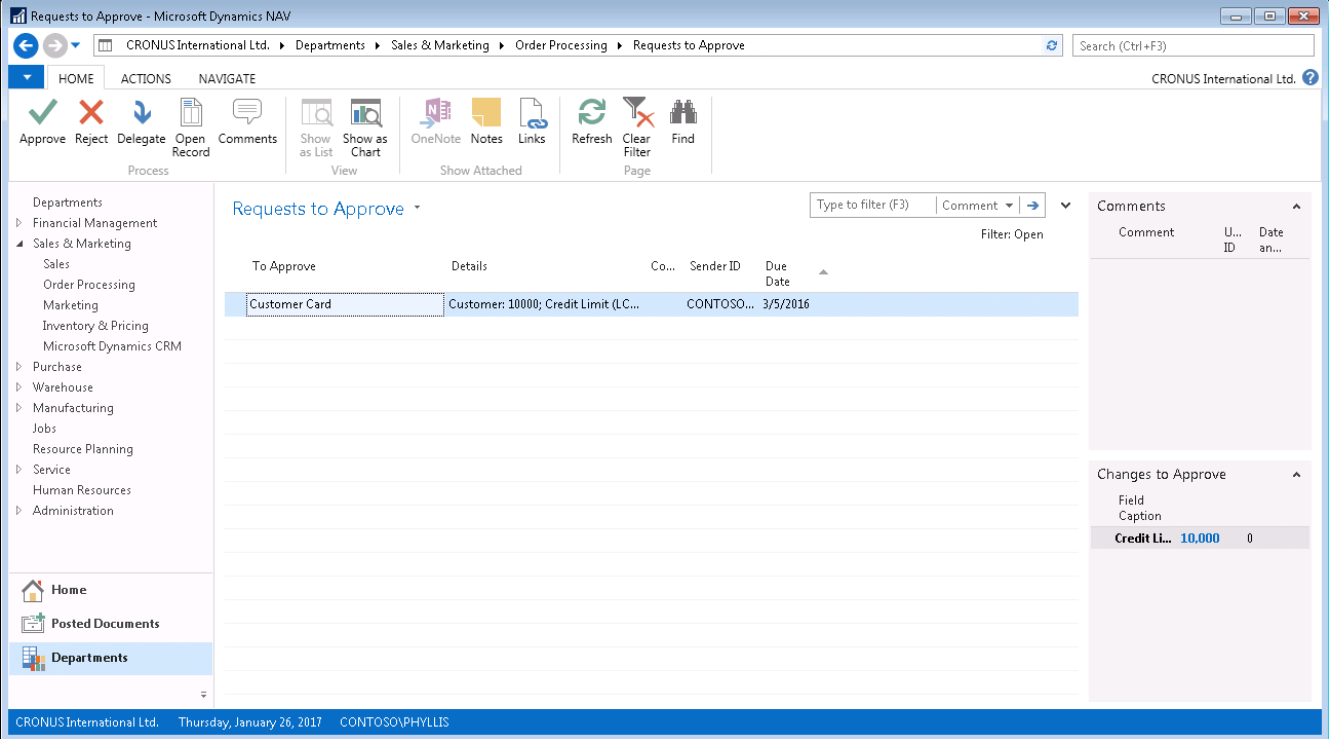
Customer Statistics

Customer No.: 10000
 Balance (LCY): 168,364.41
Sales
 Outstanding Ord...: 1,612.50
 Shipped Not Inv...: 525.50
 Outstanding Inv...: 0.00
Service
 Outstanding Ser...: 6.63
 Serv Shipped No...: 0.00
 Outstanding Ser...: 49.20
 Total (LCY): 170,558....
 Credit Limit (LCY): n 00

OK

Testing the workflow

- Logging on as Phyllis the approval request appears in the inbox
- I email notification has been set up, Phyllis have received a mail by now



Requests to Approve - Microsoft Dynamics NAV

CRONUS International Ltd. > Departments > Sales & Marketing > Order Processing > Requests to Approve

Search (Ctrl+F3)

HOME ACTIONS NAVIGATE

Process: Approve, Reject, Delegate, Open Record, Comments, Show as List, Show as Chart, View, Show Attached, OneNote, Notes, Links, Refresh, Clear Filter, Page, Find

Departments: Financial Management, Sales & Marketing (selected), Sales, Order Processing, Marketing, Inventory & Pricing, Microsoft Dynamics CRM, Purchase, Warehouse, Manufacturing, Jobs, Resource Planning, Service, Human Resources, Administration

Home, Posted Documents, Departments

Requests to Approve - Filter: Open

To Approve	Details	Co...	Sender ID	Due Date
Customer Card	Customer: 10000; Credit Limit (LC...	CONTOSO...		3/5/2016

Comments

Comment	U... ID	Date an...

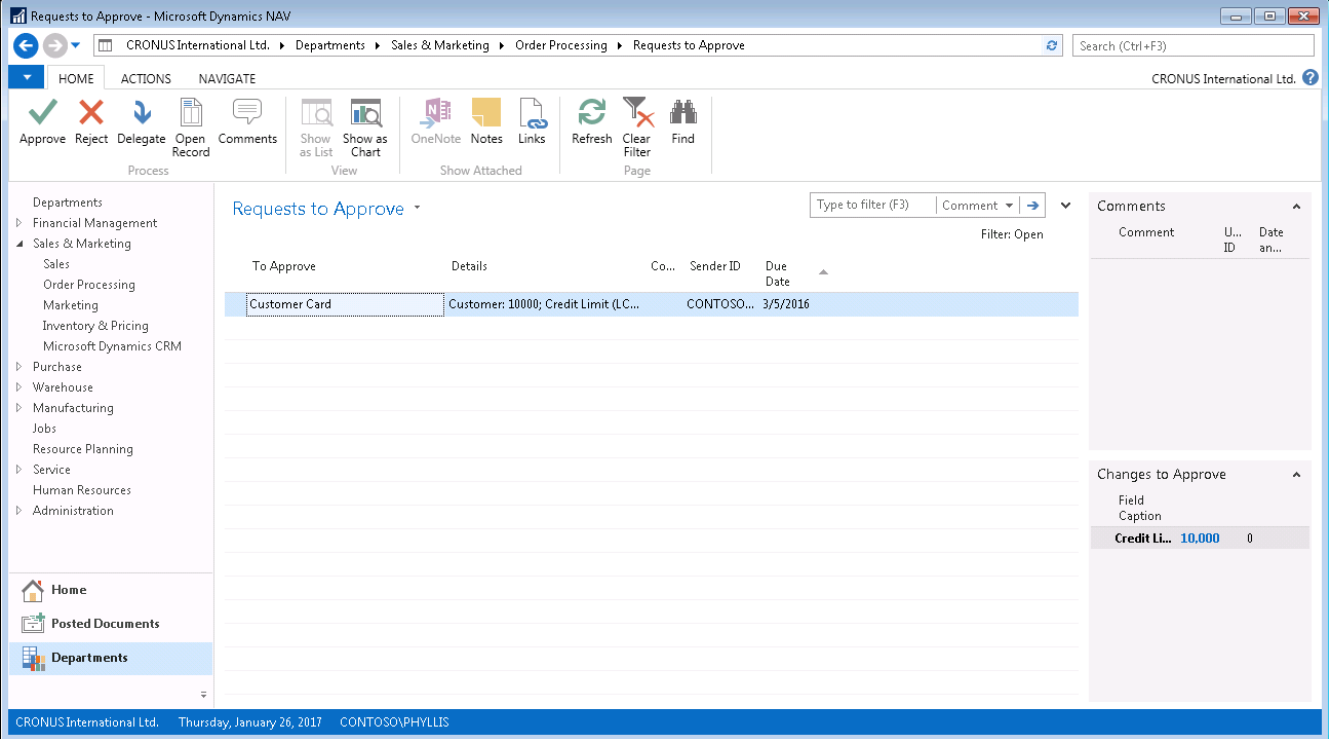
Changes to Approve

Field Caption		
Credit Li...	10,000	0

CRONUS International Ltd. Thursday, January 26, 2017 CONTOSO/PHYLLIS

Testing the workflow

- Logging on as Susan the approval request appears in the inbox
- I email notification has been set up, Susan have received a mail by now



Requests to Approve - Microsoft Dynamics NAV

CRONUS International Ltd. > Departments > Sales & Marketing > Order Processing > Requests to Approve

Search (Ctrl+F3)

CRONUS International Ltd. ?

APPROVE REJECT Delegate Open Record Comments Show as List Show as Chart OneNote Notes Links Refresh Clear Filter Page Find

Process View Show Attached Page

Departments

- Financial Management
- Sales & Marketing
 - Sales
 - Order Processing
 - Marketing
 - Inventory & Pricing
 - Microsoft Dynamics CRM
- Purchase
- Warehouse
- Manufacturing
- Jobs
- Resource Planning
- Service
- Human Resources
- Administration

Home Posted Documents Departments

Requests to Approve

Type to filter (F3) Comment Filter: Open

To Approve	Details	Co...	Sender ID	Due Date
Customer Card	Customer: 10000; Credit Limit (LC...	CONTOSO...		3/5/2016

Comments

Comment	U... ID	Date an...
---------	---------	------------

Changes to Approve

Field Caption		
Credit Li...	10,000	0

CRONUS International Ltd. Thursday, January 26, 2017 CONTOSO/PHYLLIS

Workflow architecture

- Header information
- Category: to place the workflow in the hierarchy
- Enabled: Enable the workflow

Edit - Workflow - MS-SCMAPW-01 - Sales Credit Memo Approval Workflow

HOME ACTIONS NAVIGATE CRONUS International Ltd. ?

MS-SCMAPW-01 - Sales Credit Memo Approval Workflow

Code:

Description:

Category:

Enabled:

Workflow Steps

Decrease Indent Increase Indent Delete Event Conditions New Find

When Event	On Condition	Then Response
Approval of a sales document is request...	Document Type: Credit Memo, Status: ...	(+) Add record restriction.
An approval request is approved.	Pending Approvals: 0	(+) Remove record restriction.
An approval request is approved.	Pending Approvals: >0	Send approval request for the record and create a notification.
An approval request is rejected.	<Always>	(+) Reject the approval request for the record and create a notifi...
An approval request for a sales docum...	Document Type: Credit Memo, Status: ...	(+) Cancel the approval request for the record and create a noti...
An approval request is delegated.	<Always>	Send approval request for the record and create a notification.

Workflow Responses

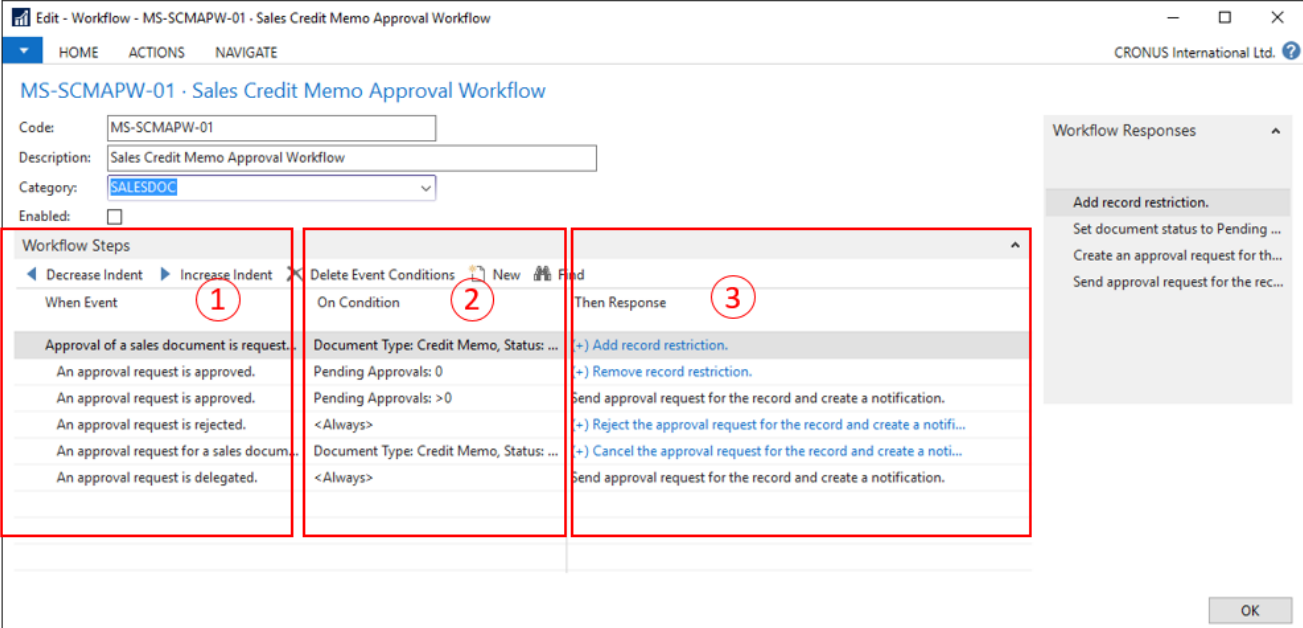
- Add record restriction.
- Set document status to Pending ...
- Create an approval request for th...
- Send approval request for the rec...

OK

Workflow architecture

Workflow steps:

- 1) Which Workflow events should be used
- 2) What should trigger the workflow
- 3) What should the response be



MS-SCMAPW-01 · Sales Credit Memo Approval Workflow

Code: MS-SCMAPW-01
Description: Sales Credit Memo Approval Workflow
Category: SALESDOC
Enabled:

Workflow Steps

When Event	On Condition	Then Response
Approval of a sales document is requested.	Document Type: Credit Memo, Status: ...	+) Add record restriction.
An approval request is approved.	Pending Approvals: 0	+) Remove record restriction.
An approval request is approved.	Pending Approvals: >0	Send approval request for the record and create a notification.
An approval request is rejected.	<Always>	+) Reject the approval request for the record and create a notification.
An approval request for a sales document is delegated.	Document Type: Credit Memo, Status: ...	+) Cancel the approval request for the record and create a notification.
An approval request is delegated.	<Always>	Send approval request for the record and create a notification.

Workflow Responses

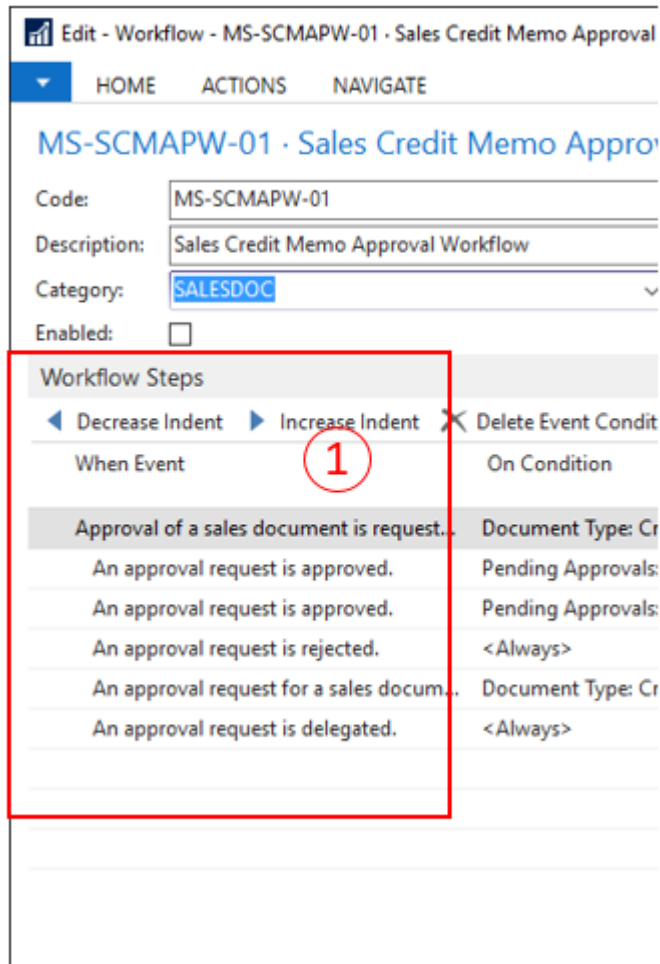
- Add record restriction.
- Set document status to Pending ...
- Create an approval request for the record and create a notification.
- Send approval request for the record and create a notification.

OK

Workflow architecture

Workflow steps:

1) Which Workflow event should be used



MS-SCMAPW-01 · Sales Credit Memo Approval

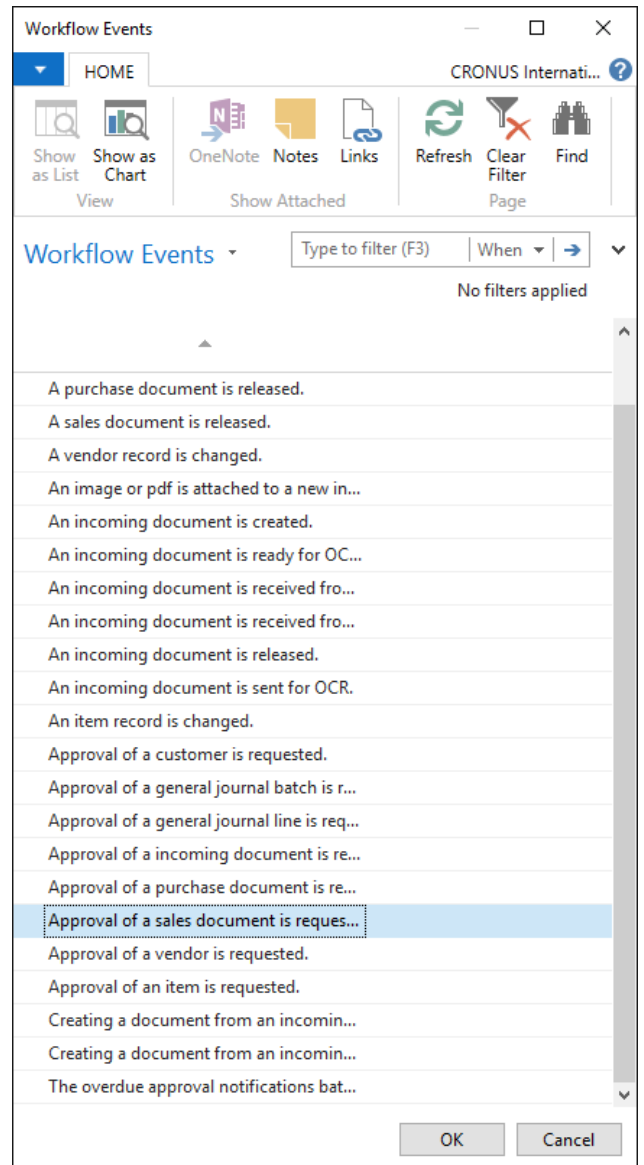
Code: MS-SCMAPW-01

Description: Sales Credit Memo Approval Workflow

Category: SALESDOC

Enabled:

Workflow Steps	
◀ Decrease Indent ▶ Increase Indent	X Delete Event Condition
When Event	On Condition
Approval of a sales document is requested...	Document Type: Credit
An approval request is approved.	Pending Approvals: Pending
An approval request is approved.	Pending Approvals: Pending
An approval request is rejected.	<Always>
An approval request for a sales document...	Document Type: Credit
An approval request is delegated.	<Always>



Workflow Events

HOME

CRONUS Internati...

Show as List Show as Chart OneNote Notes Links Refresh Clear Filter Find

View Show Attached Page

Workflow Events Type to filter (F3) When → No filters applied

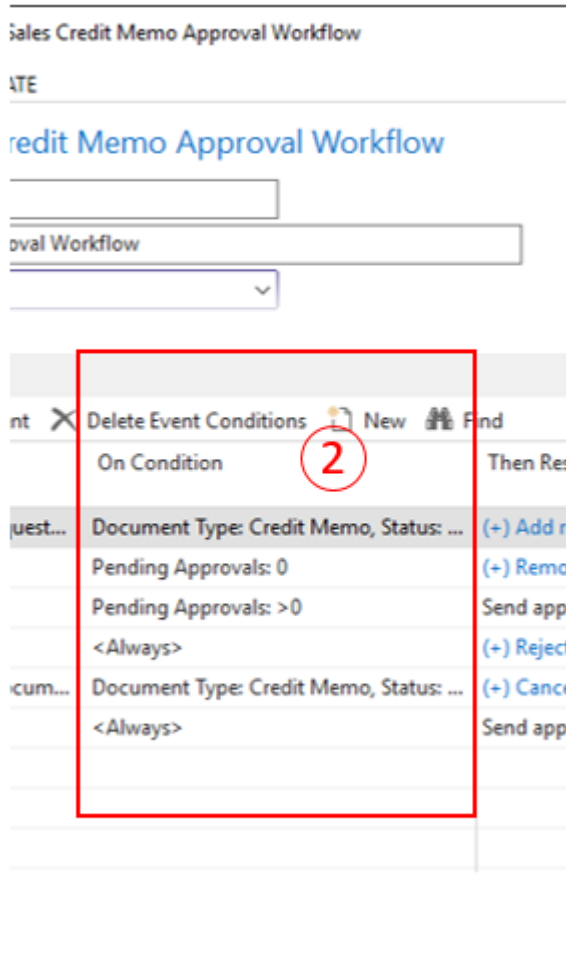
- A purchase document is released.
- A sales document is released.
- A vendor record is changed.
- An image or pdf is attached to a new in...
- An incoming document is created.
- An incoming document is ready for OC...
- An incoming document is received fro...
- An incoming document is received fro...
- An incoming document is released.
- An incoming document is sent for OCR.
- An item record is changed.
- Approval of a customer is requested.
- Approval of a general journal batch is r...
- Approval of a general journal line is req...
- Approval of a incoming document is re...
- Approval of a purchase document is re...
- Approval of a sales document is request...
- Approval of a vendor is requested.
- Approval of an item is requested.
- Creating a document from an incomin...
- Creating a document from an incomin...
- The overdue approval notifications bat...

OK Cancel

Workflow architecture

Workflow steps:

2) What should trigger the workflow



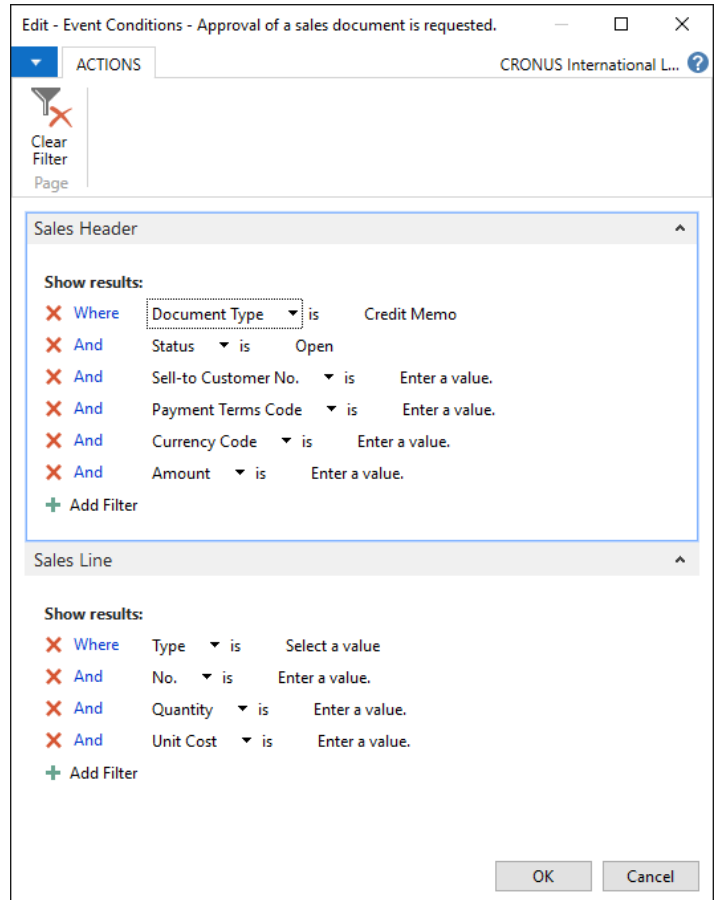
Sales Credit Memo Approval Workflow

DATE

Credit Memo Approval Workflow

Approval Workflow

Event	On Condition	Then Results
Document Type: Credit Memo, Status: ...		(+) Add r
Pending Approvals: 0		(+) Remo
Pending Approvals: >0		Send app
<Always>		(+) Reject
Document Type: Credit Memo, Status: ...		(+) Canci
<Always>		Send app



Edit - Event Conditions - Approval of a sales document is requested.

ACTIONS

CRONUS International L...

Clear Filter Page

Sales Header

Show results:

- Where Document Type is Credit Memo
- And Status is Open
- And Sell-to Customer No. is Enter a value.
- And Payment Terms Code is Enter a value.
- And Currency Code is Enter a value.
- And Amount is Enter a value.

+ Add Filter

Sales Line

Show results:

- Where Type is Select a value
- And No. is Enter a value.
- And Quantity is Enter a value.
- And Unit Cost is Enter a value.

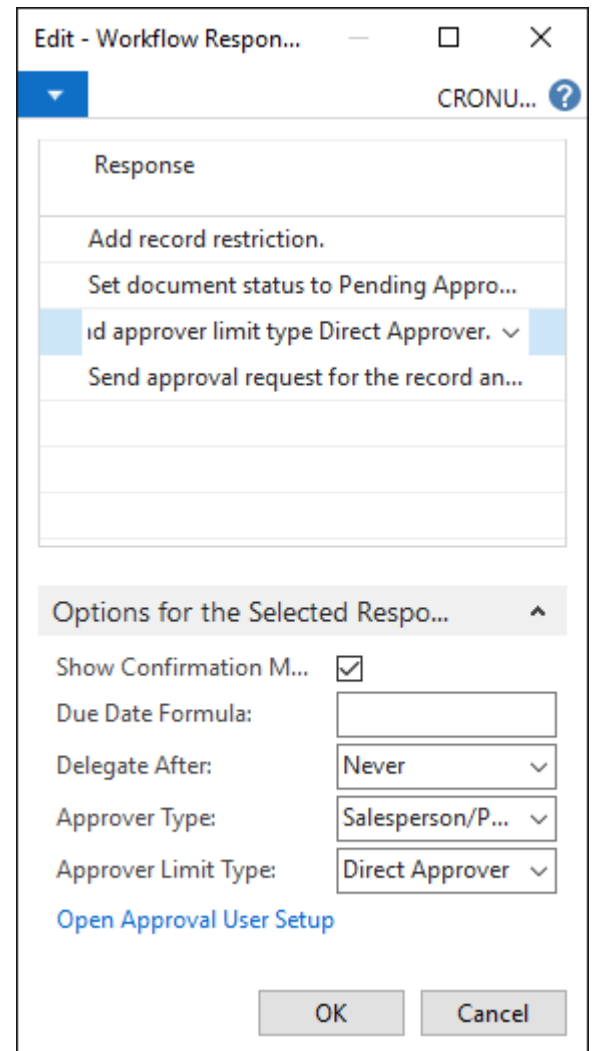
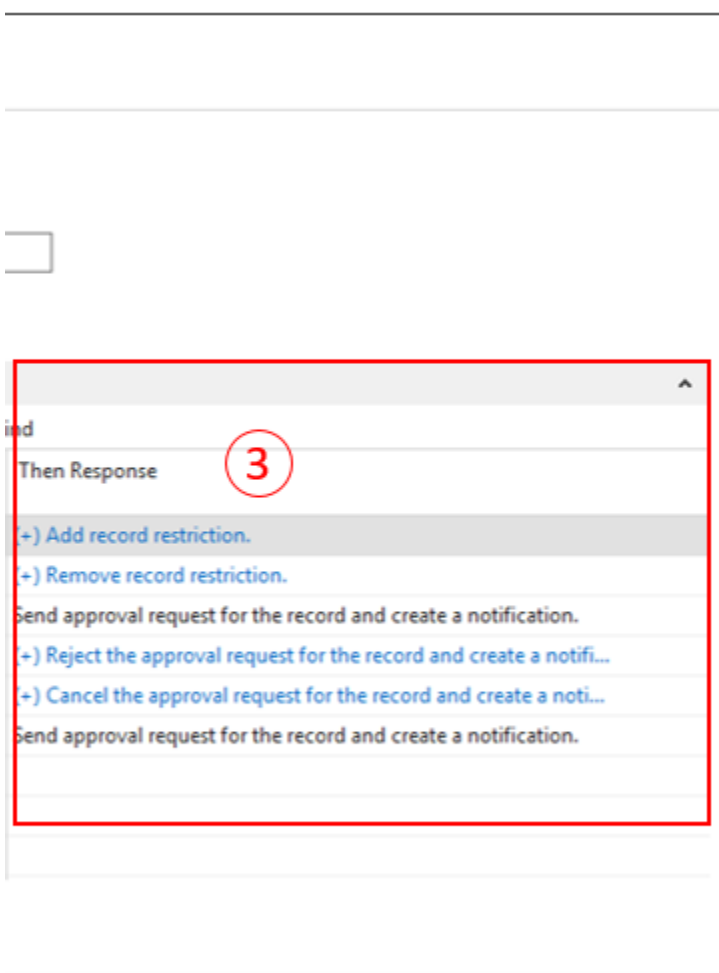
+ Add Filter

OK Cancel

Workflow architecture

Workflow steps:

3) What should the response be



Labs – Workflows

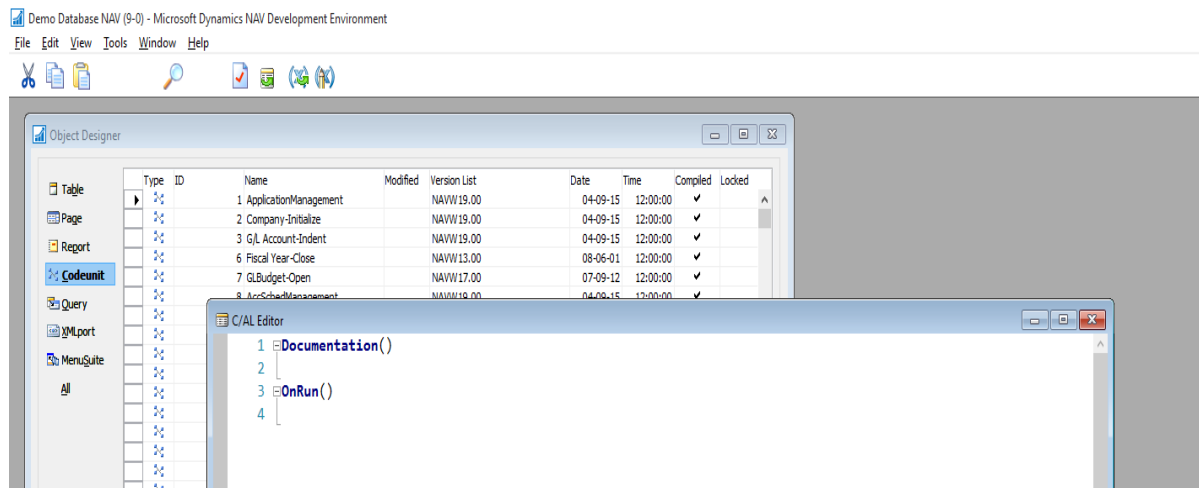
- Set up a new customer credit limit workflow from a template
- Test the workflow



Create a workflow event

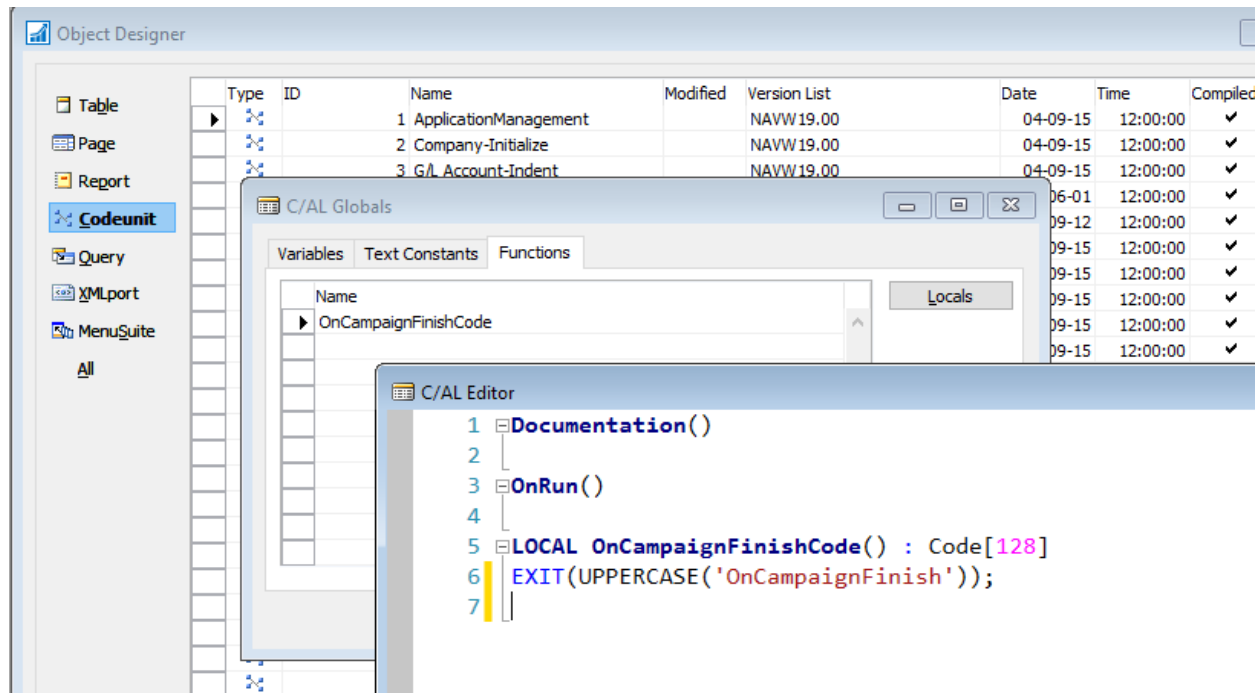
Get started by opening the Development Environment and Object Designer.

For the event we need a new codeunit.



First we'll make a helper method for binding our event with the workflow engine.

The method is a simple one which returns an identifying code.

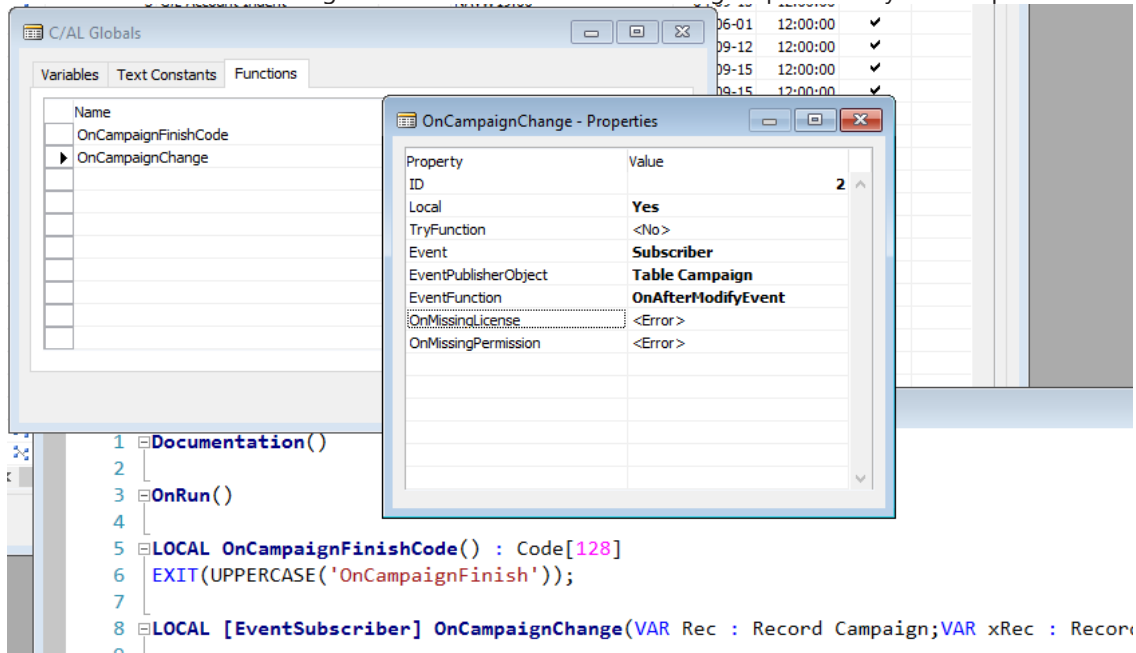


Next we add another method which will be called whenever there is a change to the Campaign table.

More specifically, this method will be called just after the **OnAfterModifyEvent** trigger for the table is called. It's the perfect time for us to catch the event and run our workflow code.

For the code itself, we'll do two things

- Show a message box so we can see our event did trigger
- Call into the workflow engine to have the NAV Workflow engine process any next steps



The screenshot shows the NAV C/AL Globals window with the 'Functions' tab selected. A list of functions is shown, with 'OnCampaignChange' selected. A 'Properties' dialog box is open for this function, showing the following configuration:

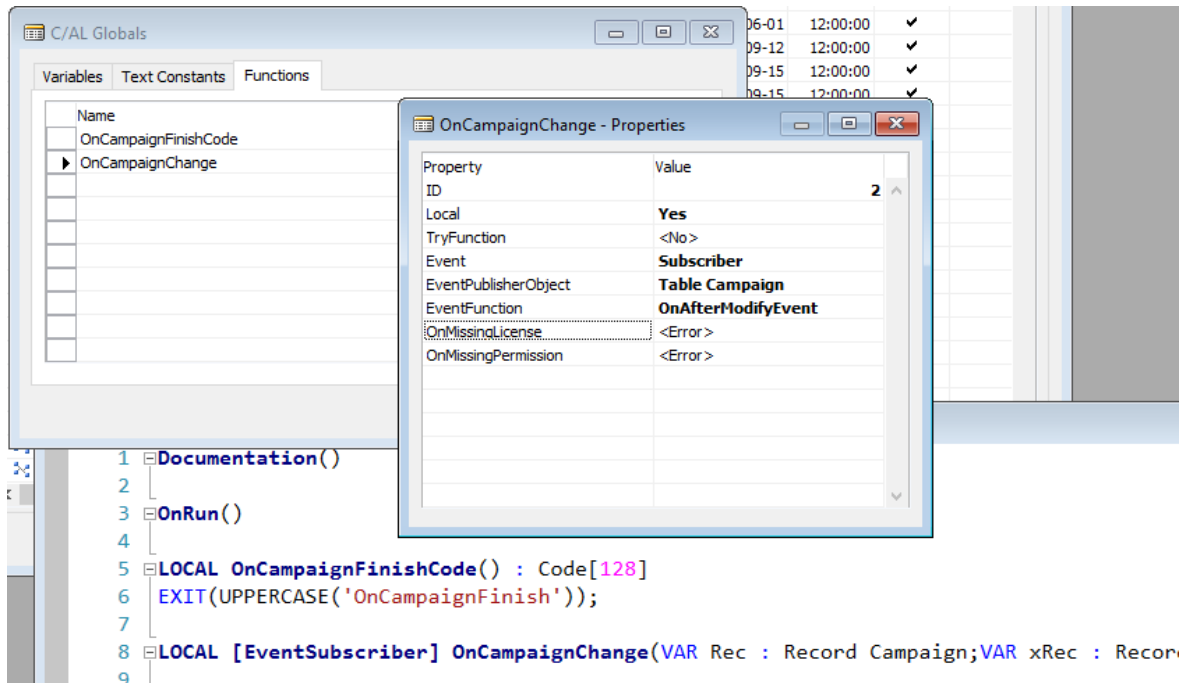
Property	Value
ID	2
Local	Yes
TryFunction	<No>
Event	Subscriber
EventPublisherObject	Table Campaign
EventFunction	OnAfterModifyEvent
OnMissingLicense	<Error>
OnMissingPermission	<Error>

The code editor below shows the following code:

```

1 Documentation()
2
3 OnRun()
4
5 LOCAL OnCampaignFinishCode() : Code[128]
6 EXIT(UPPERCASE('OnCampaignFinish'));
7
8 LOCAL [EventSubscriber] OnCampaignChange(VAR Rec : Record Campaign;VAR xRec : Record Campaign)
9

```



This screenshot is identical to the one above, showing the NAV C/AL Globals window with the 'Functions' tab selected. The 'OnCampaignChange' function is selected, and its 'Properties' dialog box is open, showing the same configuration as in the previous screenshot:

Property	Value
ID	2
Local	Yes
TryFunction	<No>
Event	Subscriber
EventPublisherObject	Table Campaign
EventFunction	OnAfterModifyEvent
OnMissingLicense	<Error>
OnMissingPermission	<Error>

The code editor below shows the following code:

```

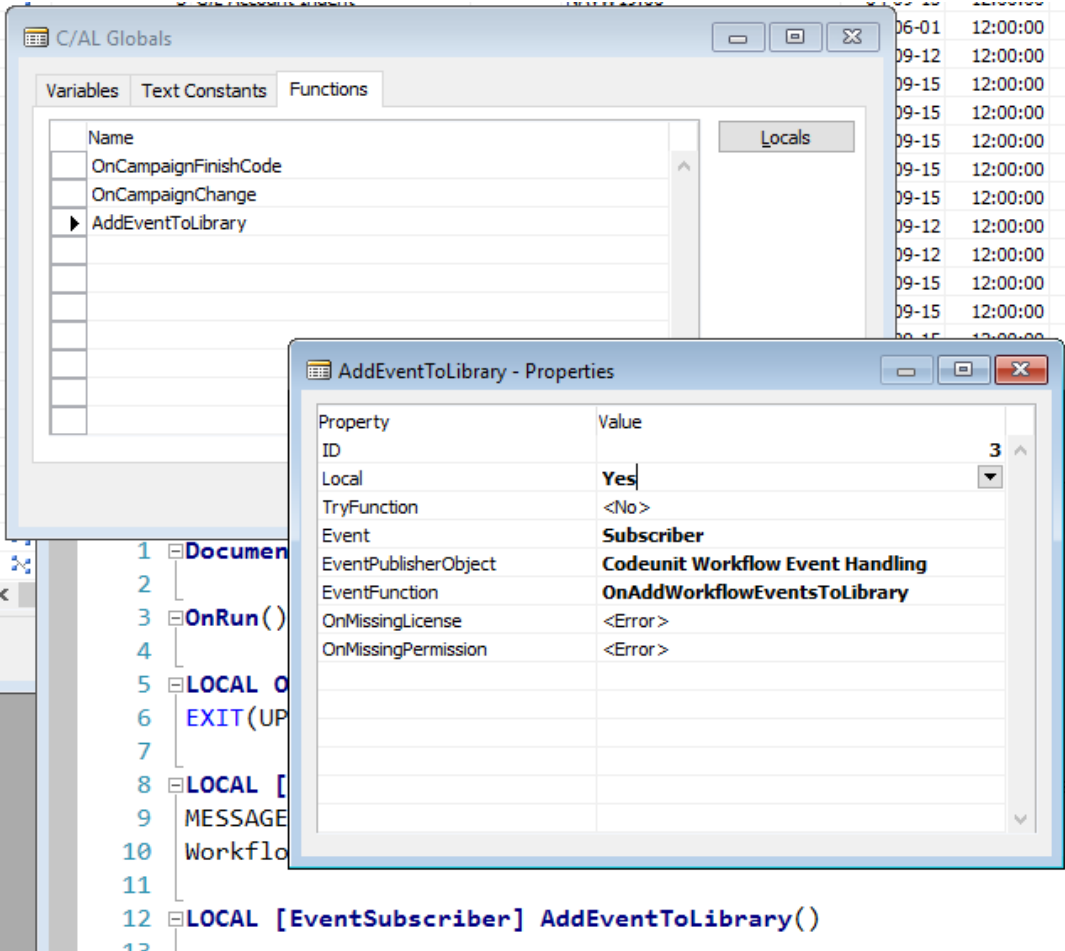
1 Documentation()
2
3 OnRun()
4
5 LOCAL OnCampaignFinishCode() : Code[128]
6 EXIT(UPPERCASE('OnCampaignFinish'));
7
8 LOCAL [EventSubscriber] OnCampaignChange(VAR Rec : Record Campaign;VAR xRec : Record Campaign)
9

```

Then we need to add another event subscriber method which is responsible for adding our event to the workflow library. The workflow library is the collection of events which can be seen and managed from the Workflow Setup page.

To add a Workflow event we need to do three things:

- Subscribe to the event.
- Define a user readable string which describes the event.
- Call a method in the workflow event handling codeunit, passing in the identifier and the descriptive string.



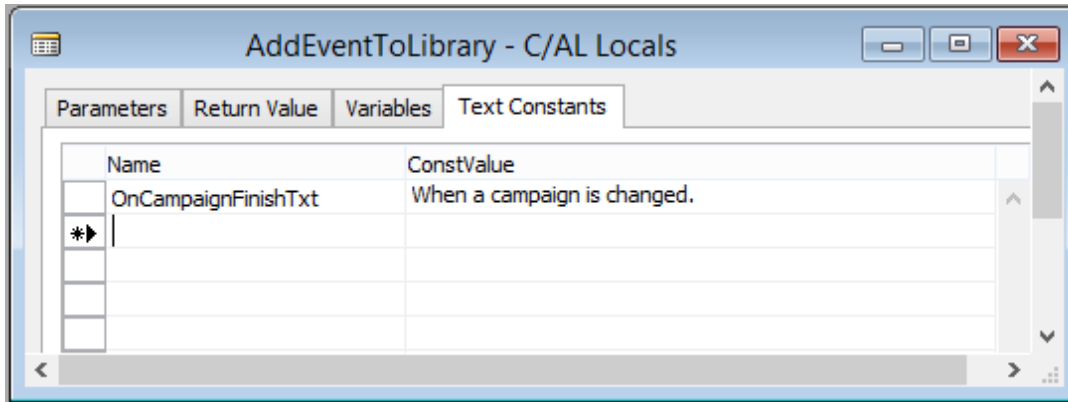
The screenshot shows two windows from the SAP development environment. The top window, titled 'C/AL Globals', has tabs for 'Variables', 'Text Constants', and 'Functions'. The 'Functions' tab is active, showing a list of functions with 'AddEventToLibrary' selected. The bottom window, titled 'AddEventToLibrary - Properties', displays a table of properties for the selected function.

Property	Value
ID	3
Local	Yes
TryFunction	<No>
Event	Subscriber
EventPublisherObject	Codeunit Workflow Event Handling
EventFunction	OnAddWorkflowEventsToLibrary
OnMissingLicense	<Error>
OnMissingPermission	<Error>

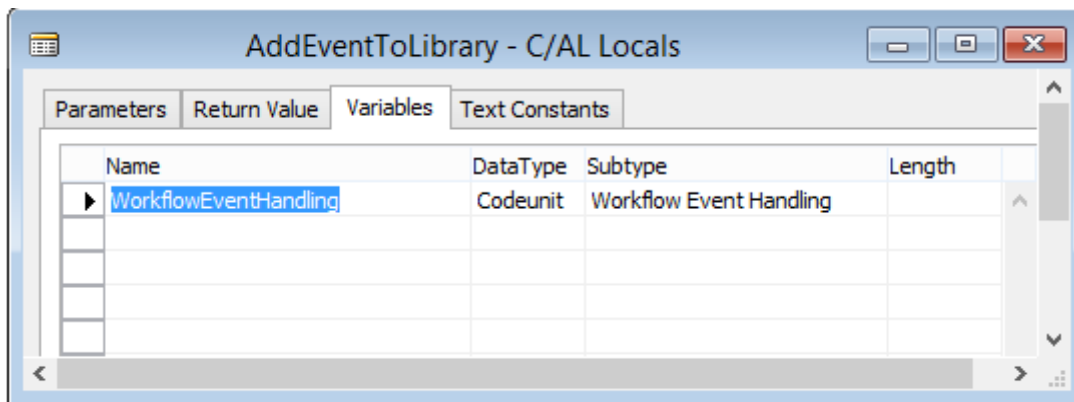
The background shows a code editor with the following code snippet:

```
1 Document
2
3 OnRun()
4
5 LOCAL O
6 EXIT(UP
7
8 LOCAL [
9 MESSAGE
10 Workflo
11
12 LOCAL [EventSubscriber] AddEventToLibrary()
13
```

And a local text constant



And a local variable



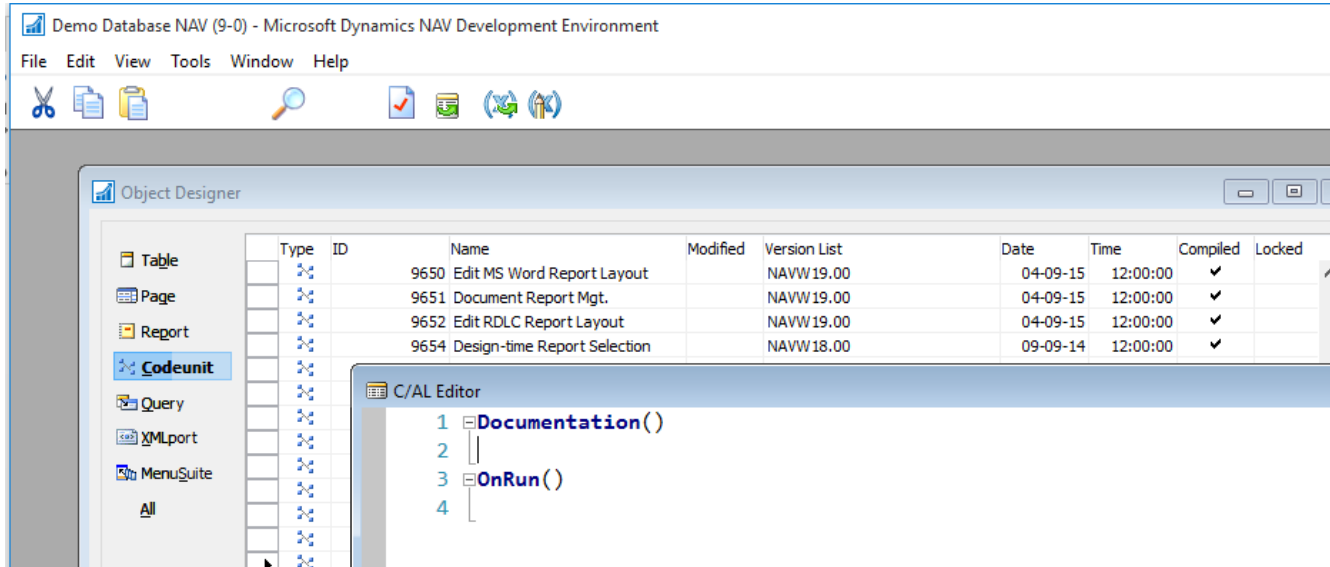
Now subscribe to the event:

```
11 |  
12 | LOCAL [EventSubscriber] AddEventToLibrary()  
13 | WorkflowEventHandling.AddEventToLibrary(OnCampaignFinishCode, DATABASE::Campaign, OnCampaignFinishTxt, 0, FALSE);  
14 |
```

Build a workflow response

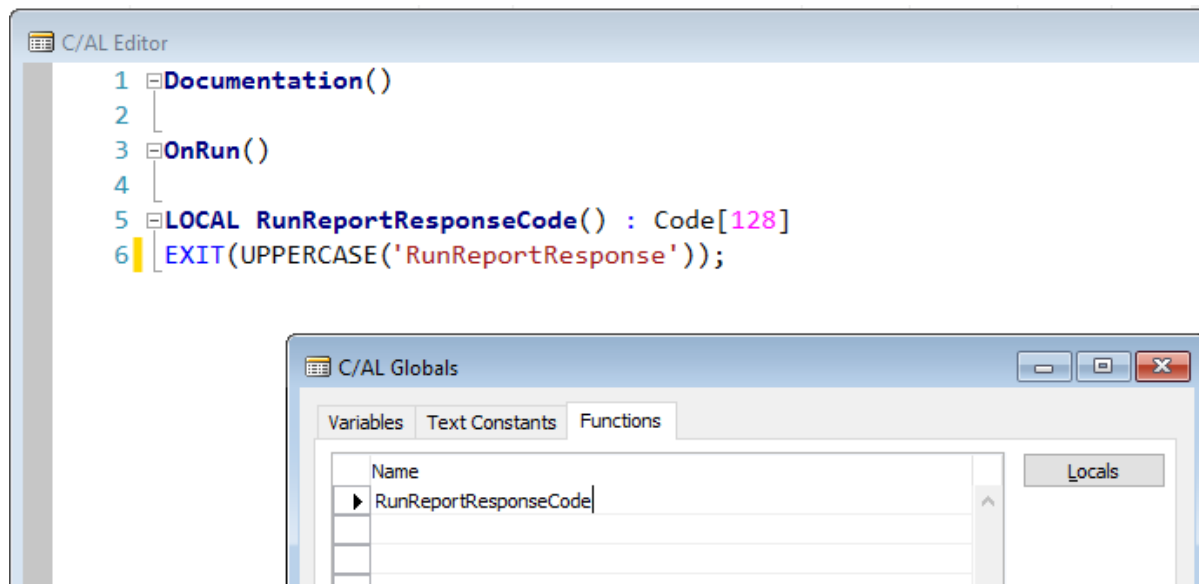
Get started by opening the Development Environment and Object Designer.

For the response we need a new codeunit.



First we'll make a helper method for binding our response with the workflow engine.

The method is a simple one which returns an identifying code.



Then we need to add another event subscriber method which is responsible for adding our response to the workflow library. The workflow library includes the collection of responses which can be seen and managed from the Workflow Setup page.

To add a Workflow response we need to do three things:

- Subscribe to the event which adds new responses to the workflow library
- Define a user readable string which describes the response
- Call a method in the workflow response handling codeunit, passing in the identifier and the descriptive string.

C/AL Editor

```

1 Documentation()
2
3 OnRun()
4
5 LOCAL RunReportResponseCode() : Code[128]
6 EXIT(UPPERCASE('RunReportResponse'));
7
8 LOCAL [EventSubscriber] AddResponseToLibrary()
9 WorkflowResponseHandling.AddResponseToLibrary(RunReportResponseCode, 0, ResponseDescription, 'Group 0');

```

AddResponseToLibrary - Properties

Property	Value
ID	2
Local	Yes
TryFunction	<No>
Event	Subscriber
EventPublisherObject	Codeunit Workflow Response Handling
EventFunction	OnAddWorkflowResponsesToLibrary
OnMissingLicense	<Error>
OnMissingPermission	<Error>

AddResponseToLibrary - C/AL Locals

Name	ConstValue
ResponseDescription	Run a report

AddResponseToLibrary - C/AL Locals

Name	Data Type	Subtype	Length
WorkflowResponseHandling	Codeunit	Workflow Response Handling	

C/AL Editor

```

1 Documentation()
2
3 OnRun()
4
5 LOCAL RunReportResponseCode() : Code[128]
6 EXIT(UPPERCASE('RunReportResponse'));
7
8 LOCAL AddResponseToLibrary()
9 WorkflowResponseHandling.AddResponseToLibrary(RunReportResponseCode, 0, ResponseDescription, 'Group 0');

```

AddResponseToLibrary - C/AL Locals

Name	ConstValue
ResponseDescription	Run a report

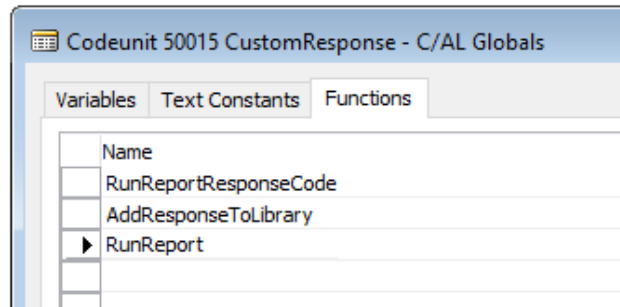
AddResponseToLibrary - C/AL Locals

Name	Data Type	Subtype	Length
WorkflowResponseHandling	Codeunit	Workflow Response Handling	

Eventually we will need the actual response code to call. We'll add this now so we can have the workflow engine invoke it when needed.

We'll write the code for it later..

```
--  
24 |  
25 |  
26 | LOCAL RunReport()  
27 | //Write code here
```

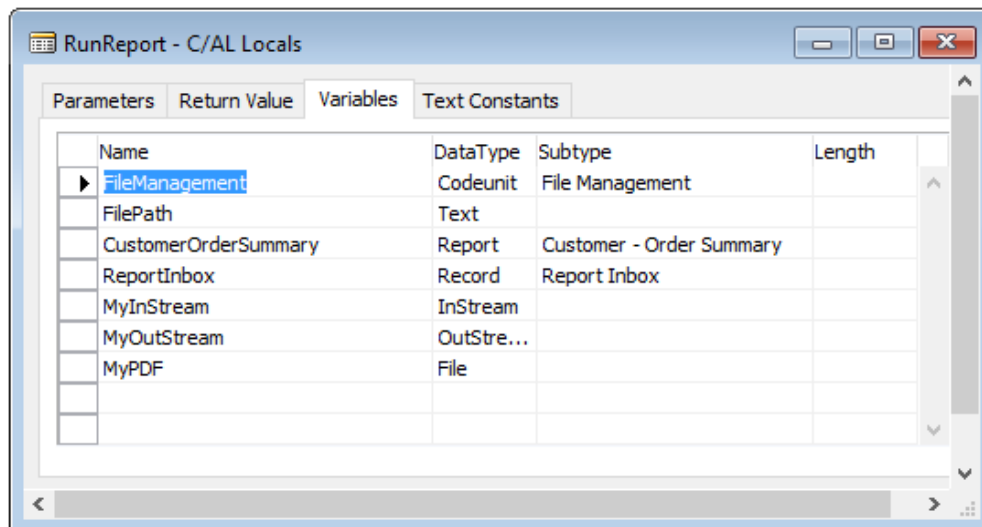
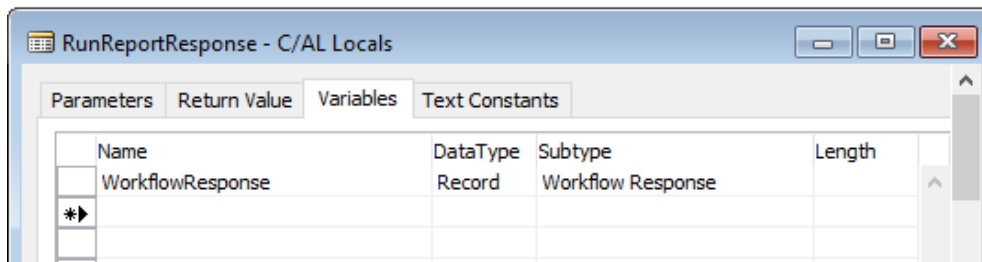


The earlier functions were used to add the response to the workflow library.

But we need another method that actually listens to when the response needs to be called.

When this code is called it needs to do two things.

Firstly it needs to evaluate if the response should be called (and it does this by checking the code value) and secondly it should return TRUE if it handles the response.





Add variables and then the code from the screenshot.

Notice that the call to RunReport and setting the Response Executed to TRUE happen only if we match to the workflow response code.

In this demo script, we'll run a specific report and save it in the report inbox. Going further, NAV Workflow allows you to build options and conditions so you can tailor what the event and response should do under specific conditions.

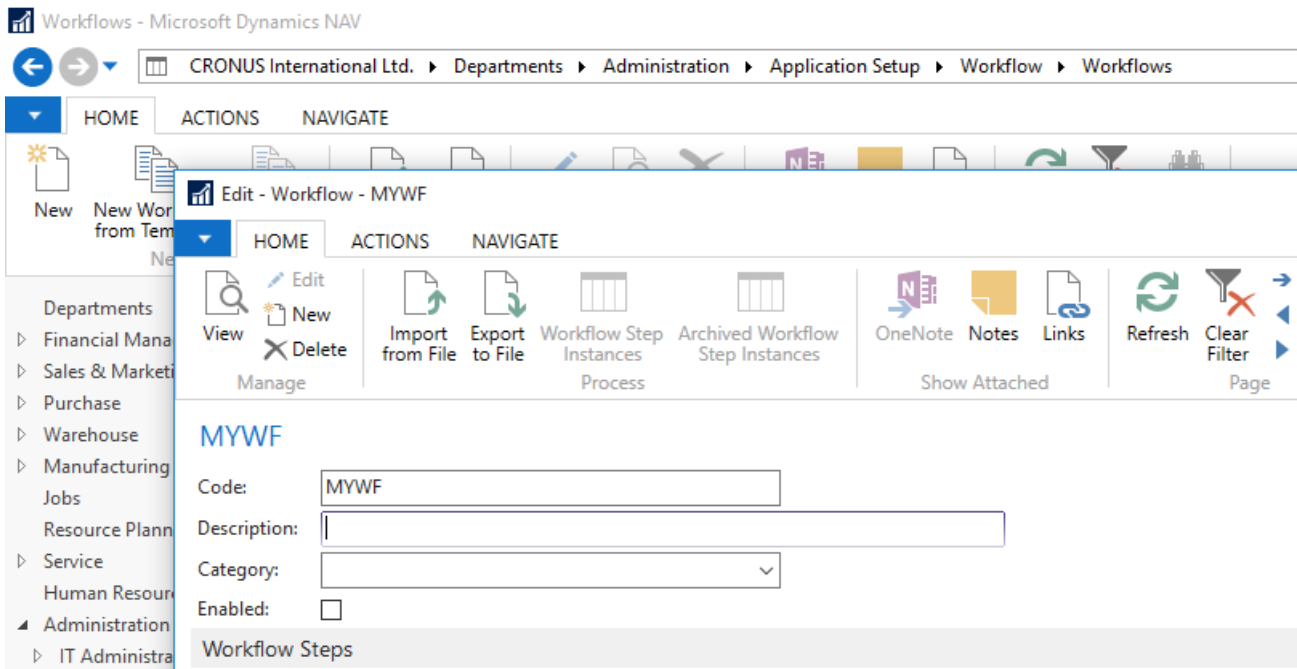
A future and improved version of this response would be to run any particular report and to specify which report and which report options should be used as part of workflow setup.

```
21 LOCAL RunReport()  
22 CustomerOrderSummary.InitializeRequest(TODAY,FALSE);  
23 CustomerOrderSummary.USEREQUESTPAGE := FALSE;  
24 FilePath := FileManagement.ServerTempFileName('.pdf');  
25 CustomerOrderSummary.SAVEASPDF(FilePath);  
26  
27 MyPDF.OPEN(FilePath);  
28 MyPDF.CREATEINSTREAM(MyInStream);  
29 ReportInbox."Report Output".CREATEOUTSTREAM(MyOutStream);  
30 COPYSTREAM(MyOutStream,MyInStream);  
31 MyPDF.CLOSE;  
32  
33 ReportInbox."User ID" := USERID;  
34 ReportInbox.Description := 'Campaign Close - Customer Order Summary';  
35 ReportInbox."Created Date-Time" := CURRENTDATETIME;  
36 ReportInbox."Report ID" := 107;  
37 ReportInbox."Report Name" := 'Customer - Order Summary';  
38 ReportInbox."Output Type" := ReportInbox."Output Type"::PDF;  
39 ReportInbox.INSERT;  
40
```

[Enable the workflow event in the NAV system](#)

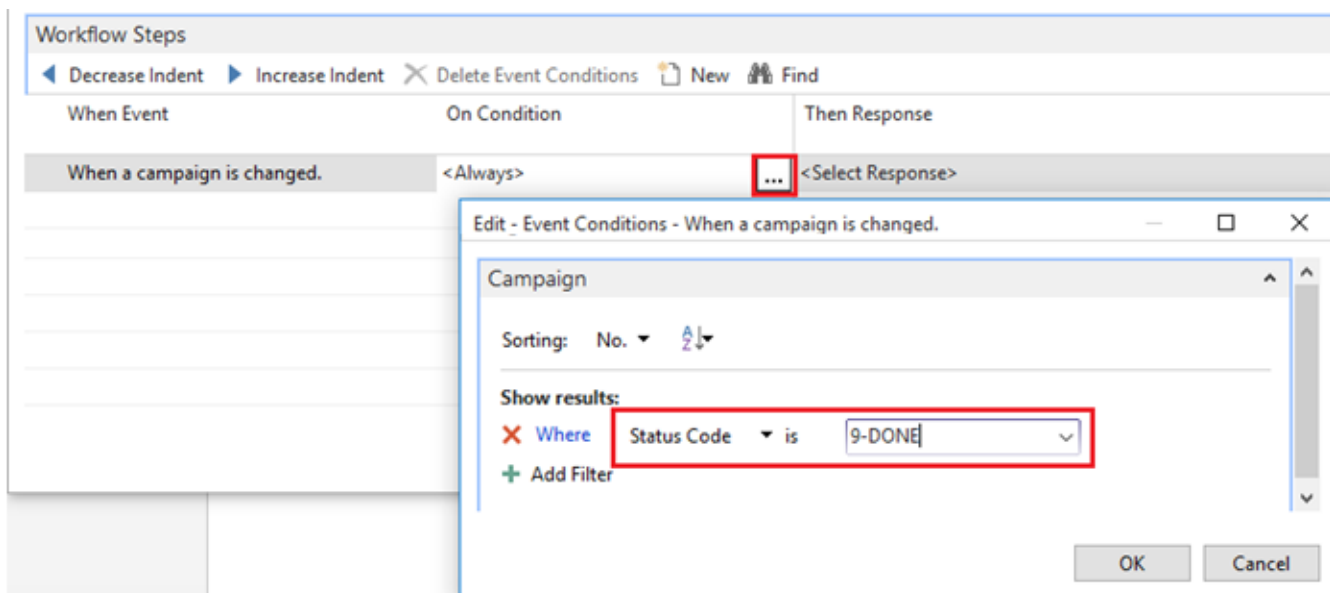
Get started by opening the client and searching for the workflow management page.

We make a new workflow and give it an identifier.

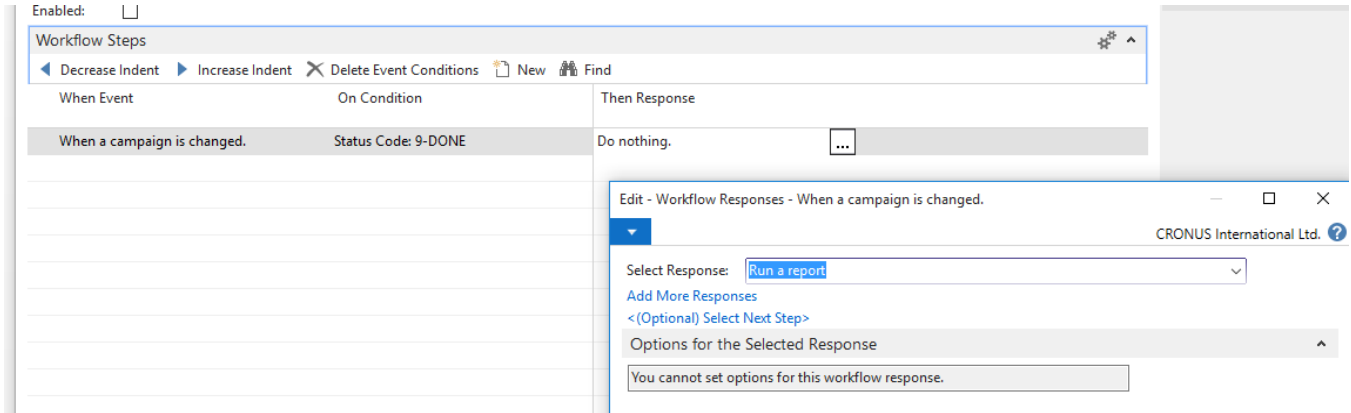


Workflows in Dynamics NAV follow a pattern we call “When, Then”. The meaning is to think about workflows as “When something happens, Then do something”. The *Whens* are Events and the *Thens* are responses.

Using the workflow steps grid, we build the steps of our workflow. In this case we can use Dynamics NAV’s drop down picker to select our new workflow event. We can recognize it from the description we coded earlier. After picking the event, we select under what conditions it should run. Here we want to only fire events when the status of a Campaign is Closed.

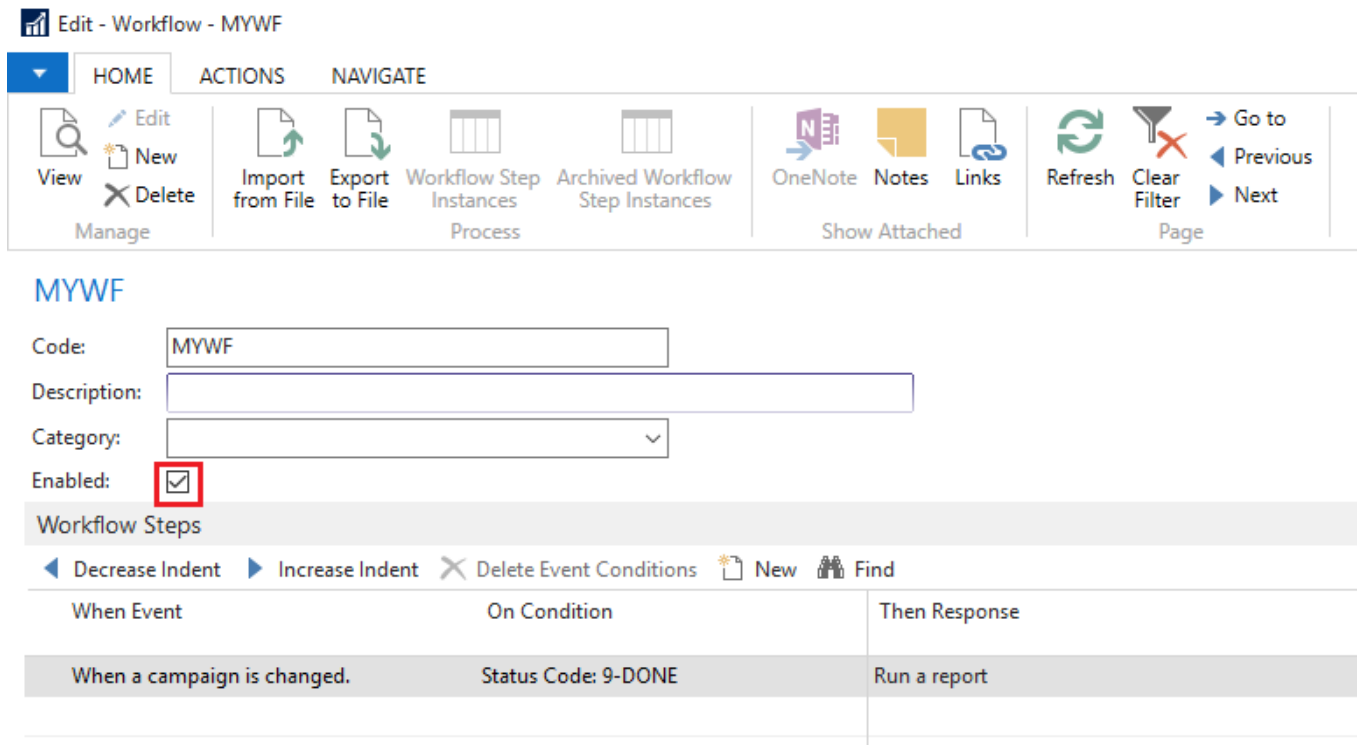


After picking the Event, we select a response. In this case we'll pick "Run a Report" because all we want to do is see that it works.



This now represents a simple workflow we can build.

Clicking 'Enabled' will activate the workflow and it will be live in our system.



Code:

Description:

Category:

Enabled:

When Event	On Condition	Then Response
When a campaign is changed.	Status Code: 9-DONE	Run a report

To test if our workflow works, we'll jump to the Campaign page and edit one of the campaigns, setting the value to Done and confirm the change.

When we do this, we see the dialog box showing that our event fired and was run from the Workflow Engine.

Edit - Campaign Card - CP1001 - Increase sale

HOME ACTIONS NAVIGATE REPORT

View Manage Process Report Campaign History Prices Send To Page Show Attached Page

CP1001 · Increase sale

General

No.:	CP1001	Ending Date:	01-04-2017
Description:	Increase sale	Salesperson Co.:	DC
Status Code:	9-DONE	Last Date Modif.:	
Starting Date:	01-11-2016	Activated:	No

Invoicing

Department C... SALES

Microsoft Dynamics NAV

Event fired on Table Campaign

OK

Notes

Click I

Now we want to see the report itself.

We'll jump back to the Role Center and look in the report inbox.

Notice in the report inbox fact box there is a new entry.

From here we can see the report we just ran and open it!

Trailing Sales Orders

All Orders(Month/No. of Orders), Updated at 14:48:19

Month	Released	Pending Prepayment	Pending Approval	Open
Jun 2017	~15	~10	~10	~10
Jul 2017	~15	~10	~10	~10
Aug 2017	~15	~10	~10	~10
Sep 2017	~15	~10	~10	~10
Oct 2017	~15	~10	~10	~10

Report Index

Created Date-Time	Description	Output Type
15-09-2015 14:48	Campaign Close - Customer Order Summary	PDF

Click Open on the file dialog and the PDF file will open!

Customer - Order Summary
CRONUS International Ltd. 15. september 2015
Page 1
EUROPE:IGGLASSON

Outstanding Orders

Customer No.	Name	..before	15-09-15	15-10-15	15-11-15	after...	Total
		14-10-15	14-11-15	14-12-15			
01454545	New Concepts Furniture	USD	0,00	0,00	0,00	1.260,54	1.260,54
10000	The Cannon Group PLC	USD	0,00	0,00	0,00	1.290,00	1.290,00
20000	Selangoran Ltd.	USD	0,00	0,00	0,00	7.481,07	7.481,07
30000	John Haddock Insurance Co.	USD	0,00	0,00	0,00	7.601,73	7.601,73
31987987	Candoxy Nederland BV	EUR	0,00	0,00	0,00	2.424,57	2.424,57
32789456	Lovaina Contractors	EUR	0,00	0,00	0,00	4.804,75	4.804,75
38128456	MEMA Ljubljana d.o.o.	EUR	0,00	0,00	0,00	176.857,92	176.857,92
40000	Deerfield Graphics Company	EUR	0,00	0,00	0,00	4.142,00	4.142,00
43687129	Designstudio Gmunden	EUR	0,00	0,00	0,00	207.323,56	207.323,56
46897889	Englunds Kontorsmbier AB	SEK	0,00	0,00	0,00	7.299,36	7.299,36
49525252	Beef House	EUR	0,00	0,00	0,00	31.800,90	31.800,90
49633663	Autohaus Mielberg KG	EUR	0,00	0,00	0,00	132.893,09	132.893,09
60000	Blanemark Hifi Shop	USD	0,00	0,00	0,00	12.520,00	12.520,00
61000	Fairway Sound	USD	0,00	0,00	0,00	1.670,00	1.670,00
62000	The Device Shop	USD	0,00	0,00	0,00	3.814,00	3.814,00
Total (LCY)			0,00	0,00	0,00	423.361,66	423.361,66

Test automation

How to start using the application test suite

Managing test suites

Test selection by churn

Failure analysis

This chapter is a compilation of available Microsoft material. More information can be found at

[https://msdn.microsoft.com/en-us/library/dn951441\(v=nav.90\).aspx](https://msdn.microsoft.com/en-us/library/dn951441(v=nav.90).aspx)

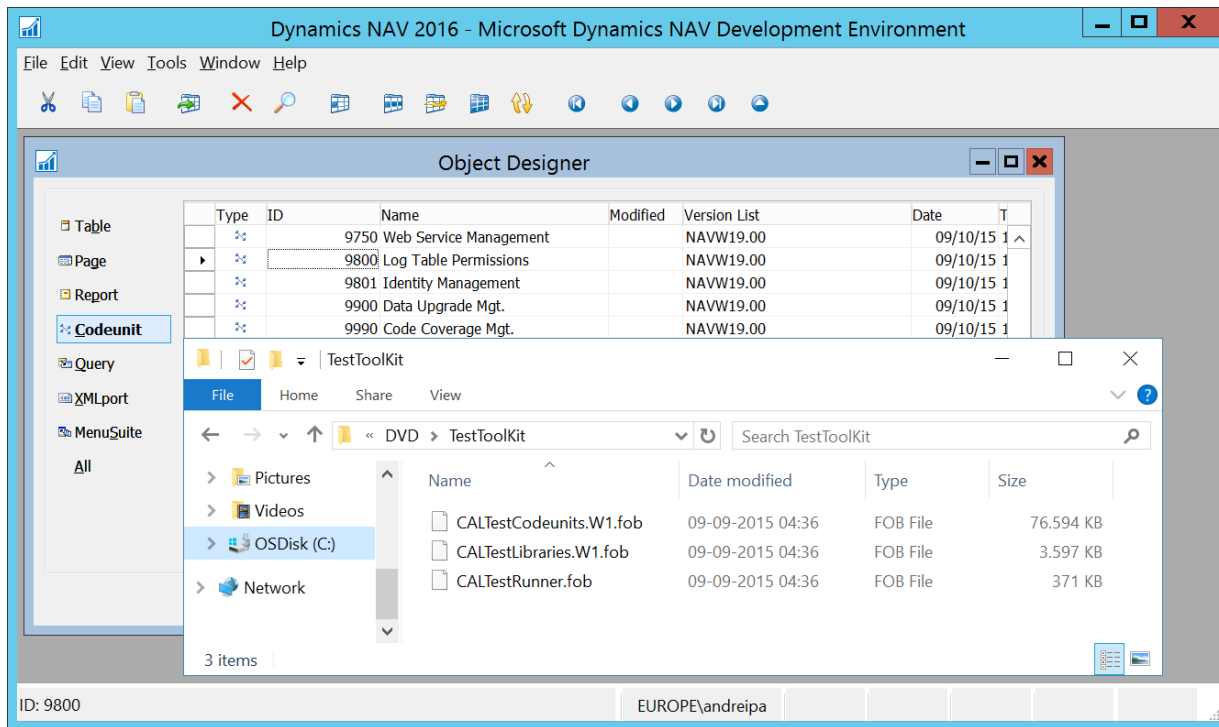


How to start using the application test suite

The Application test suite contains tools for managing and executing tests, application tests and helper functions grouped in libraries. To run tests in the CAL Test Tool window, you must import the objects in the .fob files in the TestToolkit folder that is included on the Microsoft Dynamics NAV product media.

It is enough to import CALTestCodeunits.W1.fob and CALTestLibraries.W1.fob.

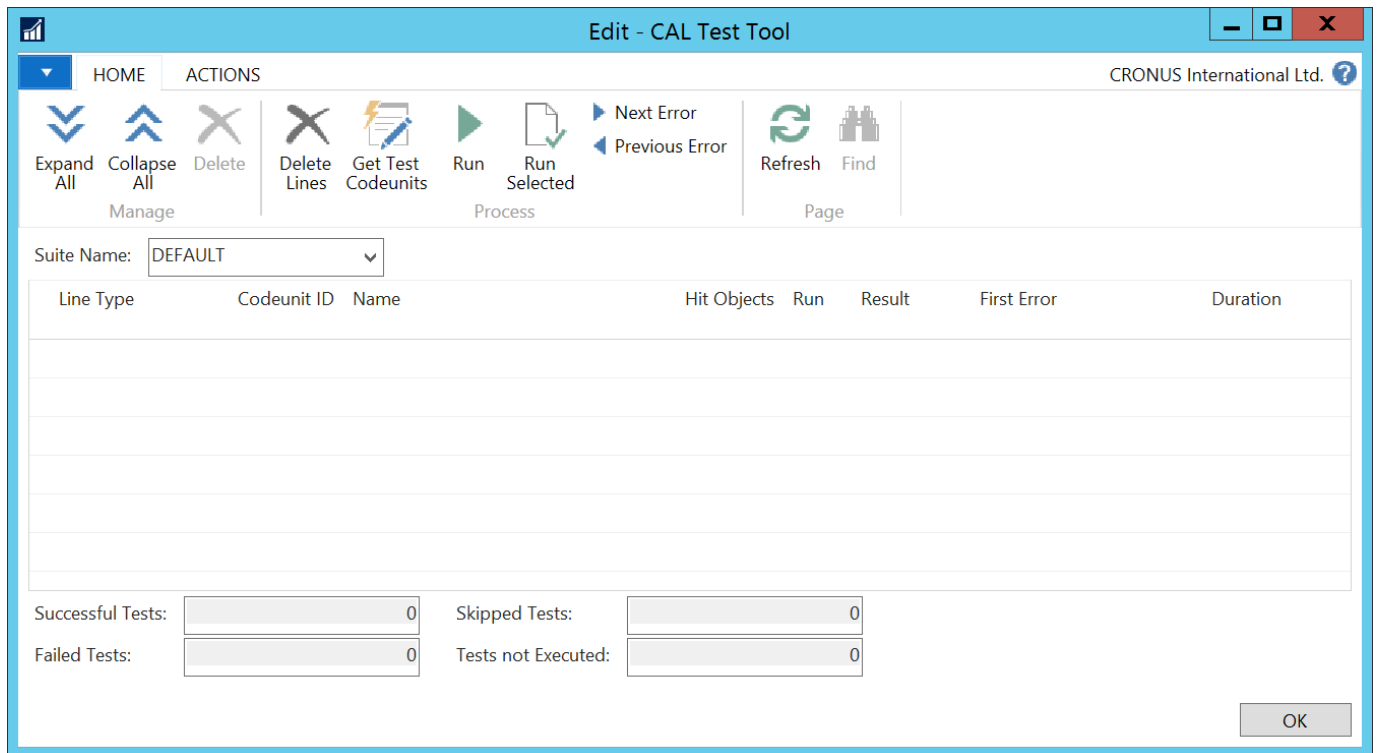
CALTestRunner.fob is already part of demo database.



Now we are ready to run the standard application tests.

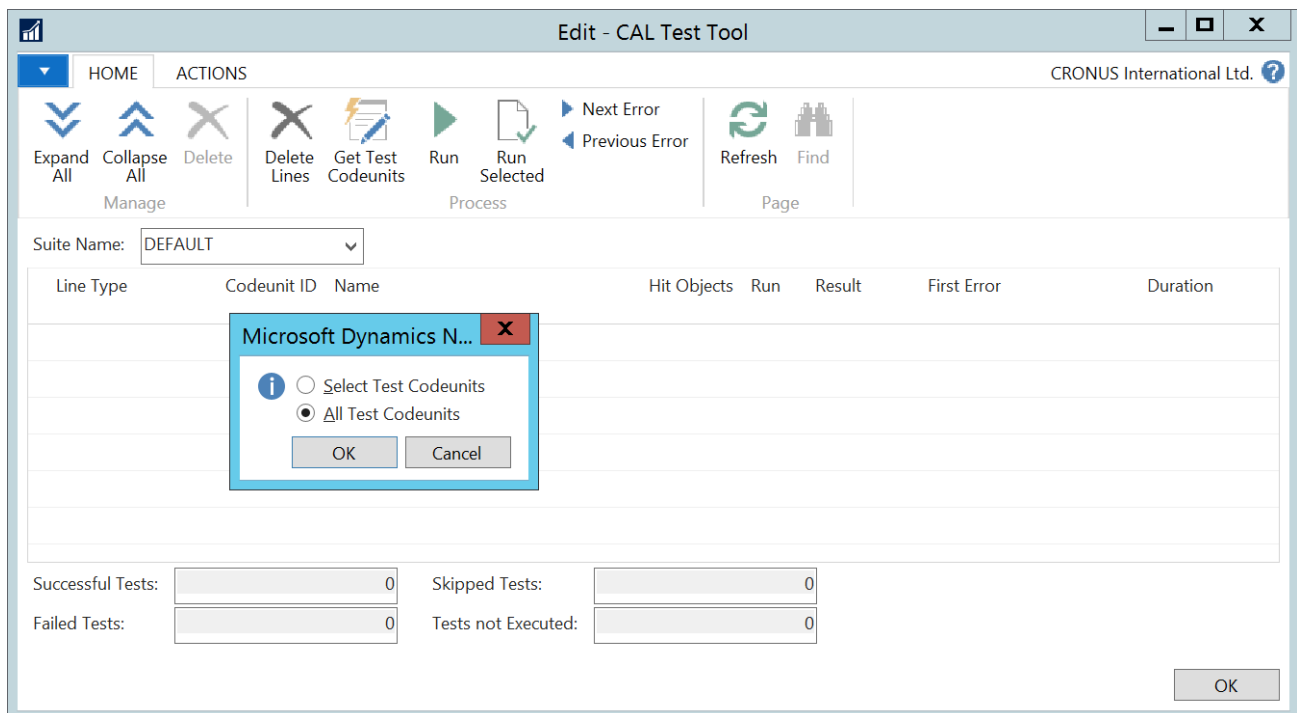
In the current demo script we will use the Test Tool for running tests. There are two options to run the Test Tool:

1. Run the CAL Test Tool page (130401) directly from the Development Environment
2. Run Test Tool from windows client.



In this demo scenario we select all available test codeunits for execution by choosing the **All Test Codeunits** option.

After Microsoft Dynamics NAV finishes loading all test codeunits, they are displayed on the **Test Tool** page.

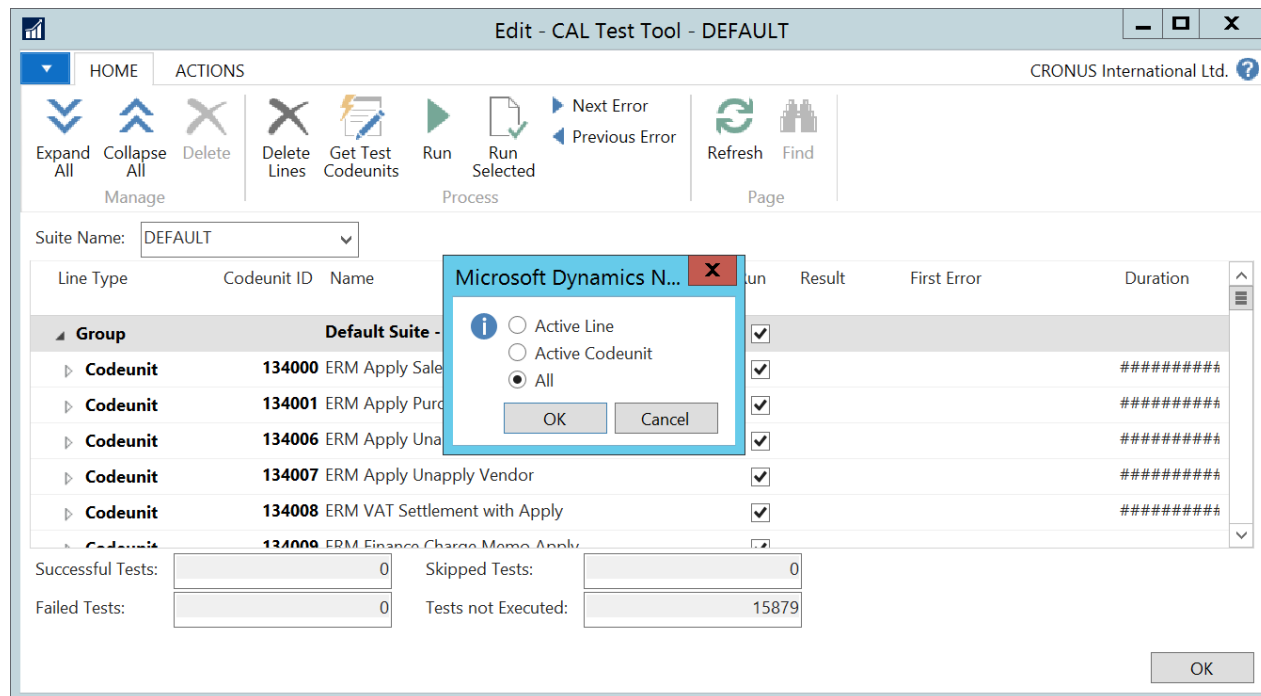


The loaded tests will take about several hours to run depending on your machine and the setup.

When the execution is completed it will show the results in the **Test Tool** page.

Alternatively, you can run individual codeunits by selecting them and clicking **Run Selected** action.

Another option is to choose either **Active Line** or **Active Codeunit** after clicking the **Run** action. **Run** field may help if needed to exclude some tests of test codeunits.

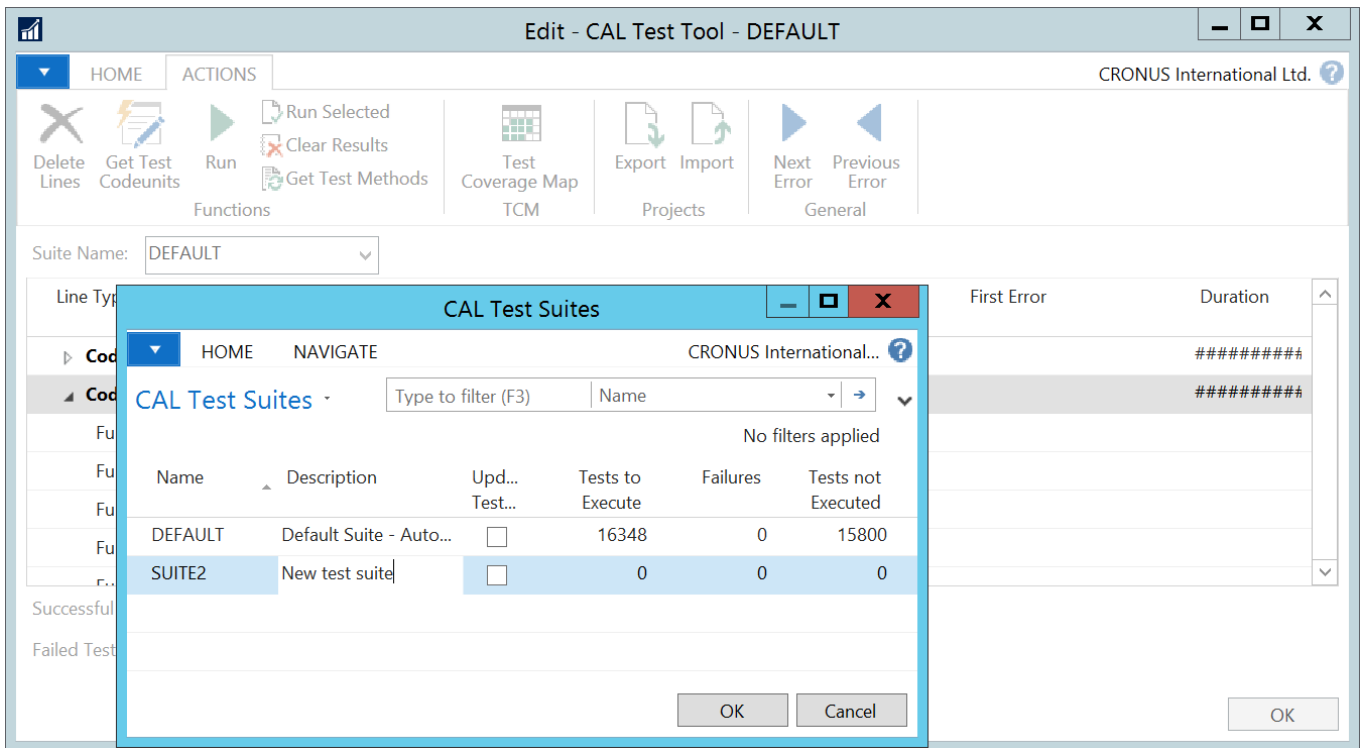


Managing test suites

The Test tool supports multiple lists of tests grouped in different test suites.

The new test suite can be created similar to journal or worksheet batches.

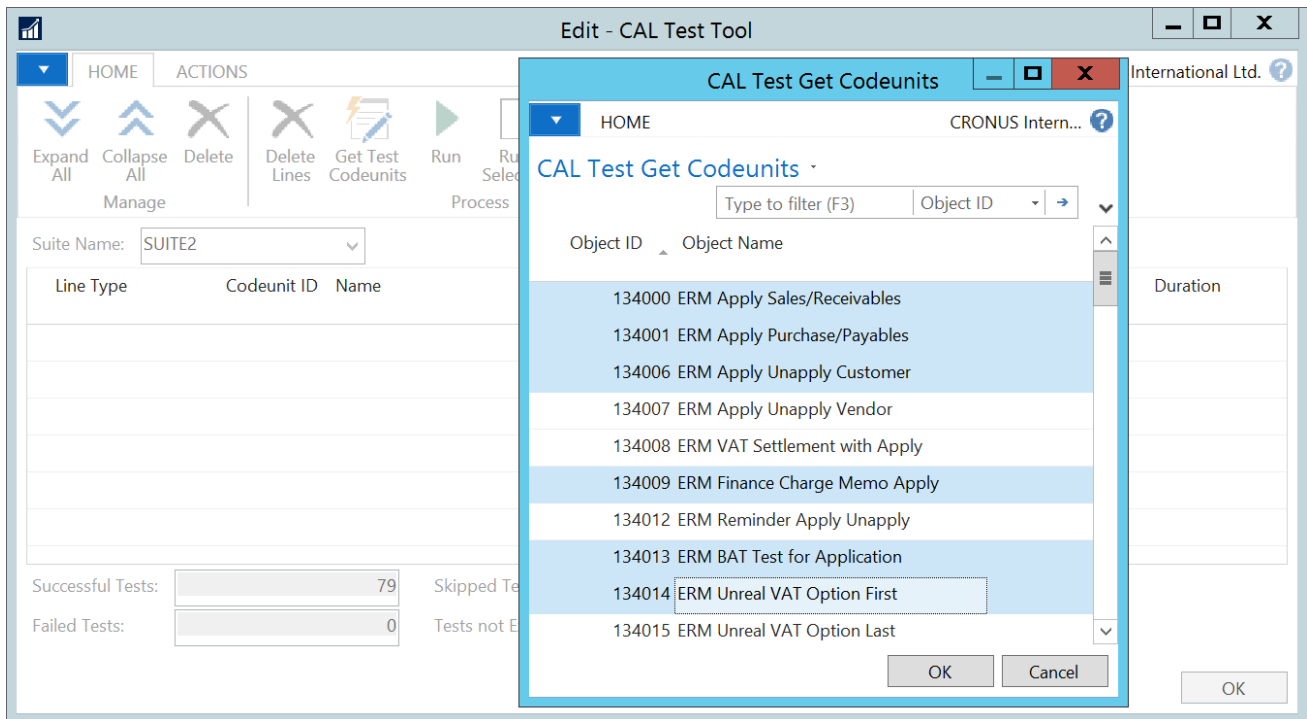
Notice the field **Update Test Coverage** map, which we will be used in next demo script.



In current scenario we will not add all available test codeunits but only the needed subset. To do this we select the option **Select Test Codeunits**.

The CAL Test Get Codeunits window will be opened. This page displays all available test codeunits, including your own.

Added codeunits can be executed as was described earlier.

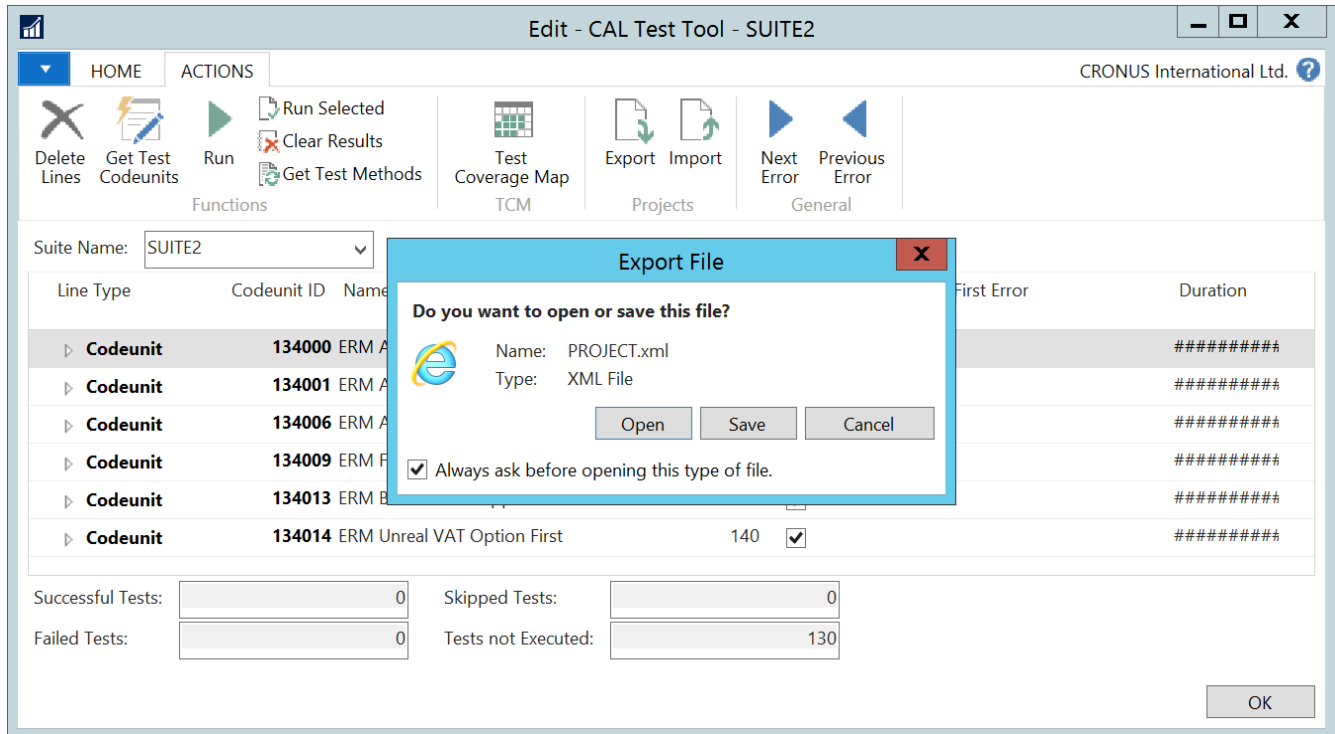


You can export the test suite into XML format and share it with your colleagues.

For example, the exported test suite might contain tests related to a specific project or modification. Or exported test suite might contain tests related to the most important or critical functionality of application and should be executed more often than other tests.

To import the tests use the **Import** action on the **Action** ribbon.

The exported test suite can be used as input for automated test execution.



Test selection by churn

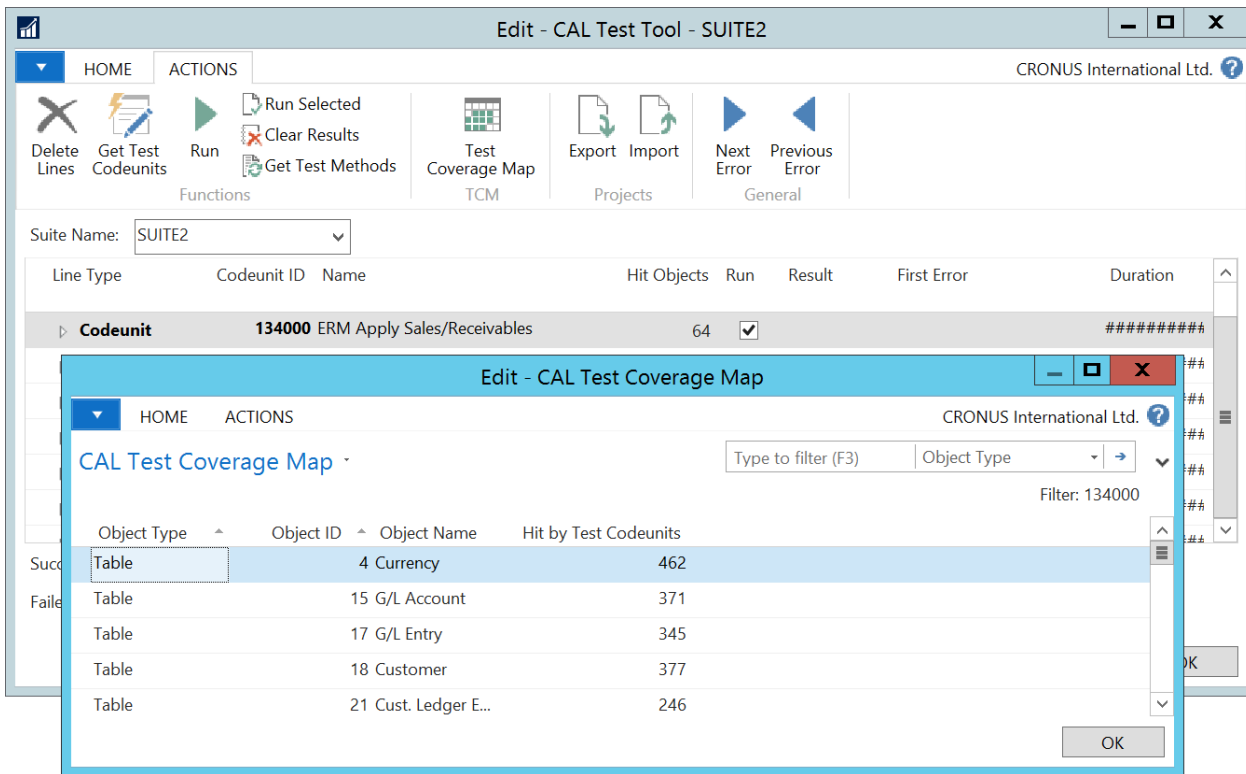
During development, it is not feasible to have to wait long to get the results from tests. Therefore we build the **Test Coverage Map** feature, which helps you narrow the set of tests down to the relevant tests.

The feature works by analyzing the code coverage data from individual test codeunits and comparing it to the set of objects that have to be tested. This is called the test coverage map and it is stored in table **130406 CAL Test Coverage Map**.

The **Test coverage map** can be created during the test execution. Activate checkbox "**Update Test Coverage Map**" on the test suite and all test executions will update the test coverage map data.

You can notice how **Hit Object** was changed. It displays related data from the **Test Coverage Map** table.

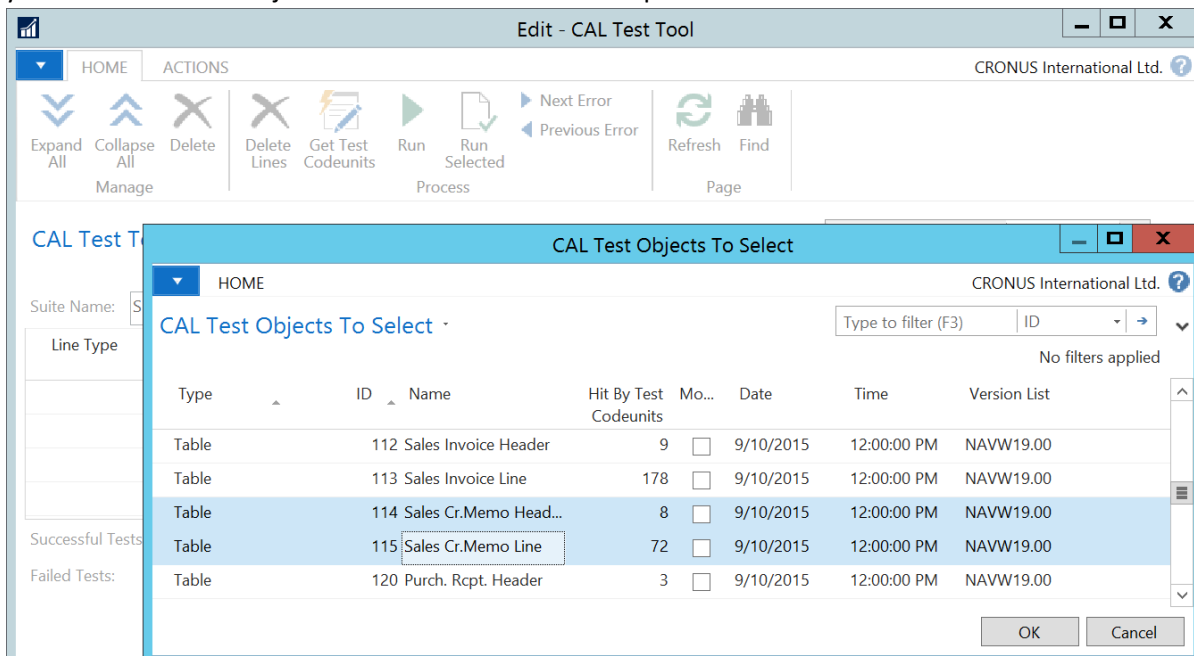
Pay attention that you can export and import the **Test Coverage Map** with the actions **Import** and **Export** located in the ribbon.



How can we benefit from the **Test Coverage Map**? For example, you merged an add-on and wants to be sure that this did not cause any regression in existing functionality. You know which objects were modified in this Add-on and you want to run a subset of tests relevant for these objects.

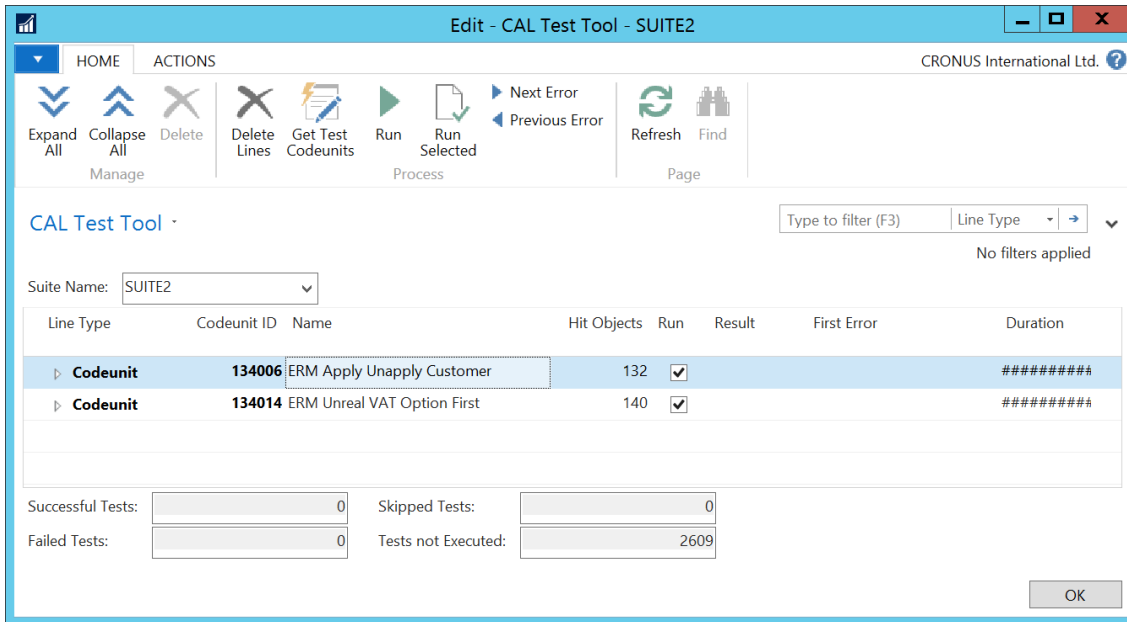
As Test Coverage Map is now available, the prompt which appears when you click **Get Test Codeunit** action, will now include two new options.

Once you select the option **Get Test Codeunits Based on Selected Objects** the system shows you the list where you can choose the objects for tests. In this demo script we will test tables related to Sales Credit Memo.





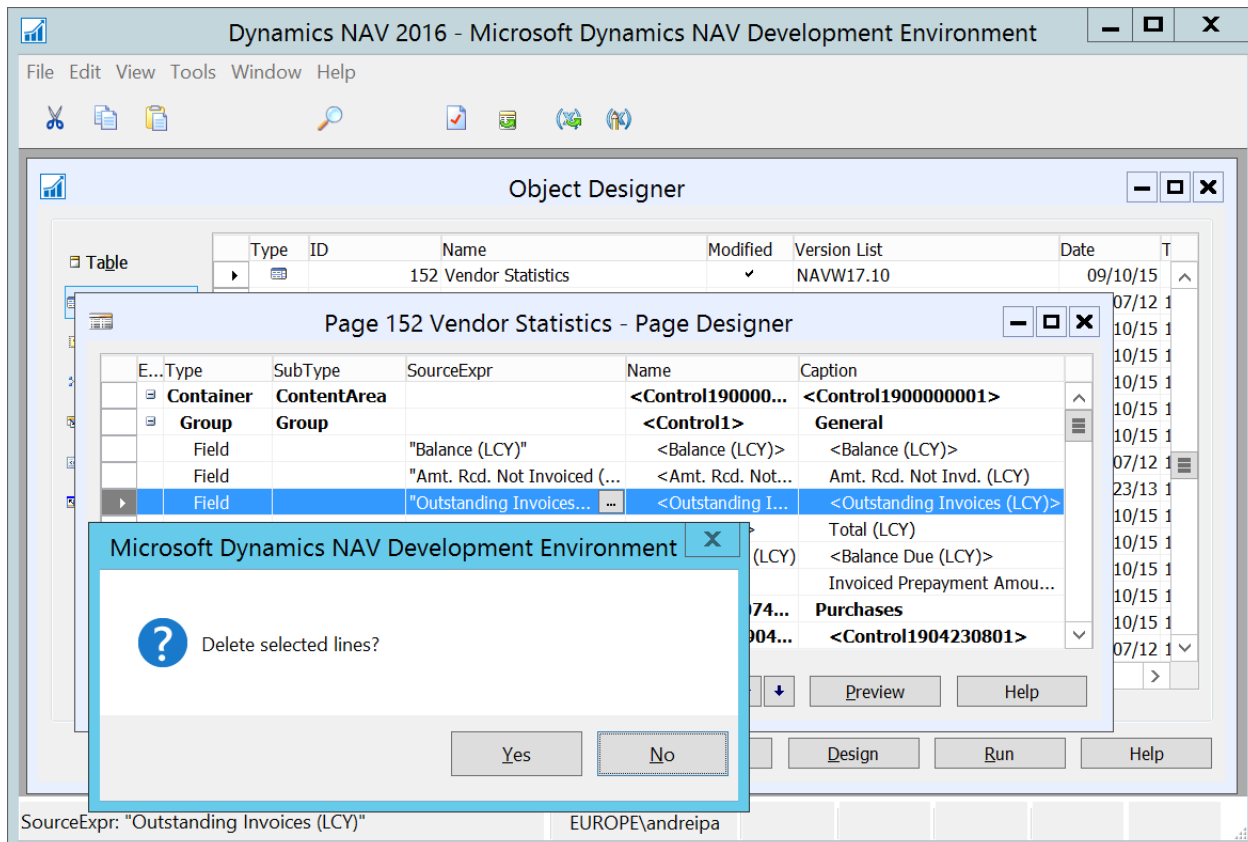
Only two test codeunits were added. The execution of two codeunits is much faster than the execution of hundreds.



Failure analysis

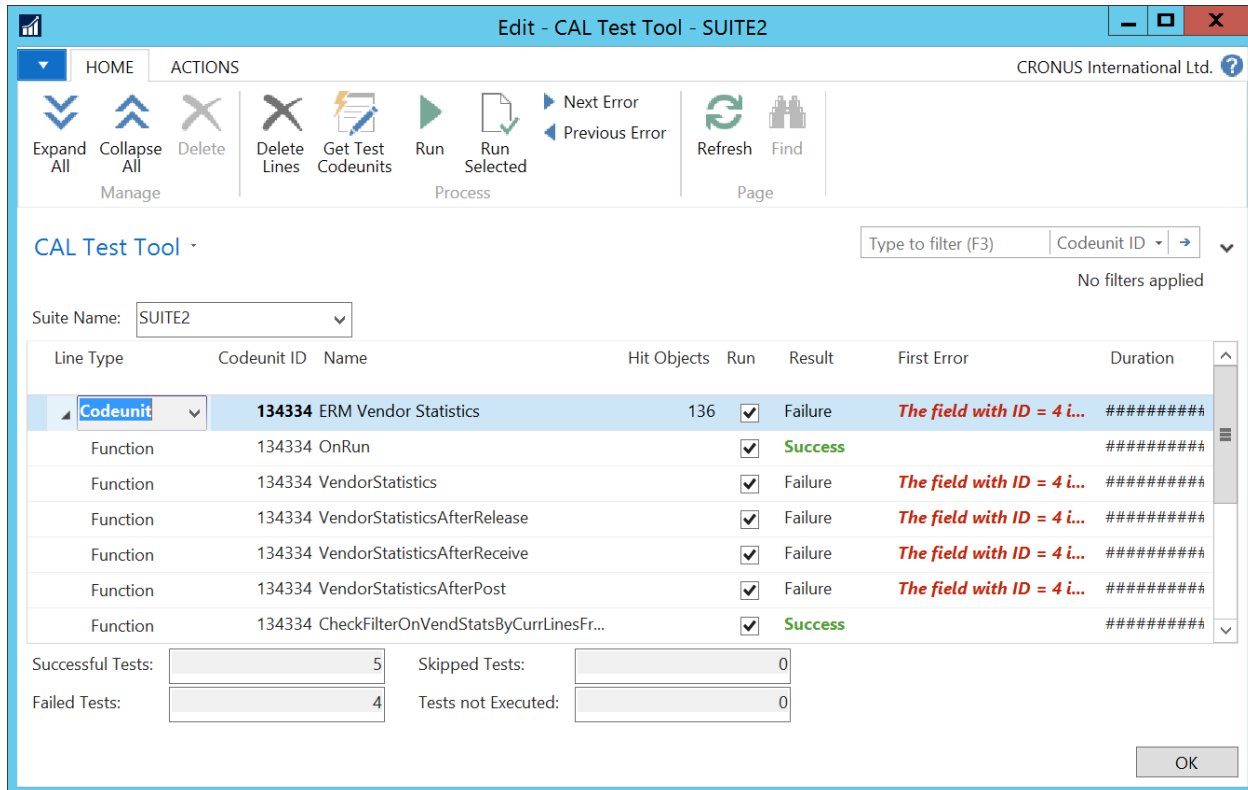
In this demo the developer did some modifications which affects standard functionality.

For example, on customer's request he deleted the "Outstanding Orders (LCY)" field from **Vendor Statistics** page.





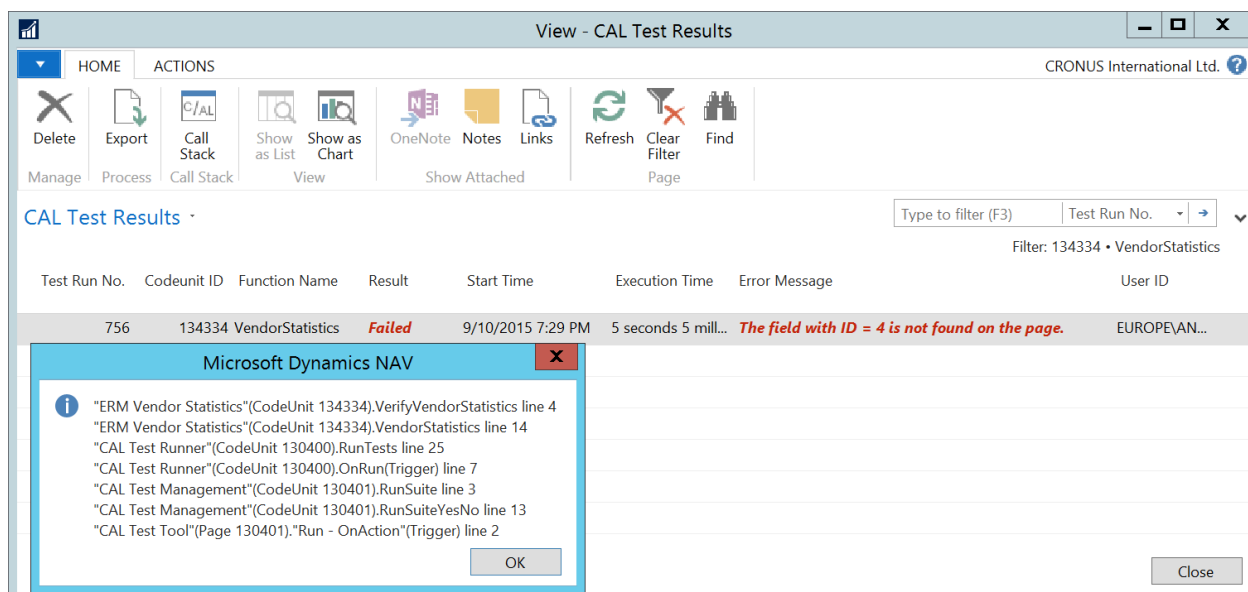
The developer wants to be sure that his modification didn't cause regressions in standard functionality. With help of the Test Coverage Map he finds out that for modified object exists only one test codeunit. As result of the test run, he discovered that 4 of 9 tests failed.



In the window CAL Test Result it's possible to see test results with additional details.

This page allows a developer to see the call stack, which help understand the reason of failure or at least helps to find out place to set break point.

Make sure that test results can be exported in XML format and shared with other developers.



Designing pages for the Universal App

Designing for Different Screen Sizes on Tablet and Phone

Differences and Limitations When Developing Pages for the Microsoft Dynamics NAV Universal App

How to: Implement the Camera in C/AL

How to: Implement Location in C/AL

This chapter is a compilation of available Microsoft material. More information can be found at

[https://msdn.microsoft.com/en-us/library/dn757143\(v=nav.90\).aspx](https://msdn.microsoft.com/en-us/library/dn757143(v=nav.90).aspx)



Designing for Different Screen Sizes on Tablet and Phone

When designing application pages for the Microsoft Dynamics NAV Tablet client and the Microsoft Dynamics NAV Phone client, it is best practice to consider the size of the tablets or phones that your end-users have access to. It is an advantage if the solution works well on both small screen sizes and also on larger screen sizes, but we also recommend that you consider thoroughly the most frequently used screen sizes for your end-user experience. Designing for small screens can be a challenge, because pages will show fewer fields, columns, and tiles.

In many cases, end-users will have access to a broad range of devices having different screen sizes and resolutions. End-users may have one device at work, and a different device at home, and expect the experience on both devices to be equally good. A good way to identify issues on how your application pages will look is to test on the smallest supported screen size. Currently tablet sizes start in the 7" range and phones range from 3" up to just below 7". There are certain requirements for running the Microsoft Dynamics NAV Universal App on tablets and phones.

Microsoft Dynamics NAV platforms



Form Factor Considerations

Users can scroll the content area of the Microsoft Dynamics NAV Universal App on a tablet to access all data for a given page. However, some elements of the screen, for example, the app bar cannot be scrolled. The app bar is the blue area of Microsoft Dynamics NAV Universal App and it is designed to provide easy access to important information and tasks that the user should not lose sight of when scrolling. The static elements will display only as much data as they can reasonably fit on the screen. Developers should design to make sure that the important static elements are displayed first so that these will be shown even on the smallest, available devices.



On phones there is no app bar, and the Microsoft Dynamics NAV Universal App displays only one part at a time on the Role Center. On the Home page, the Activity tiles are always displayed first, and you navigate through the top menu to explore the content area.

Guidance for Page Element Types on Smallest Tablet Devices

The following table provides a list of non-scrollable elements in the page content or the app bar.

Page Type	Displays on smallest tablet device
RoleCenter	4 tiles in 1 group, or 2 groups together with 2 tiles
List Pages	5 columns of type Text50 or 8 columns of type Text20
Card Pages	CardPage Factbox with up to 15 fields 2 CardPage Factboxes with up to 6 fields each Activities Factboxes with 4 tiles in 1 group, or 2 groups together with 2 tiles
Document Pages	CardPage Factbox with up to 15 fields 2 CardPage Factboxes with up to 6 fields each Activities Factboxes with 4 tiles in 1 group, or 2 groups together with 2 tiles

Testing Using a Browser

Using a browser you can test how your application pages will look on various device sizes. When running Microsoft Dynamics NAV Tablet client or Microsoft Dynamics NAV Phone client in a browser, you can use Microsoft Edge Developer Tools to emulate different screen sizes.

Differences and Limitations When Developing Pages for the Microsoft Dynamics NAV Universal App

Developing for the Microsoft Dynamics NAV Tablet client and Microsoft Dynamics NAV Phone client is not much different from developing for the Microsoft Dynamics NAV Windows client, or Microsoft Dynamics NAV Web client. There are, however, some natural limitations on tablets and phones, such as not having a physical keyboard and mouse, as well as a smaller screen. In addition to this, there are some differences and limitations in developing pages for Microsoft Dynamics NAV Tablet client and Microsoft Dynamics NAV Phone client. The differences and limitations listed are additional to the differences and limitations that exist on the Microsoft Dynamics NAV Windows client and Microsoft Dynamics NAV Web client.

Differences and Limitations Overview

The following table describes some of the most common differences and limitations that you might experience when developing for Microsoft Dynamics NAV Tablet client and Microsoft Dynamics NAV Phone client.

Concept	On Tablet	On Phone	Example	Recommendation
Activity buttons	Only the Home activity button is shown.	Only the Home activity button is shown.	Home, Departments, and Posted Documents on the Sales Order Processor Role Center.	Design pages to expose the workflows needed by the user. For example, configure the profile to show the important list pages under the Home activity button. Alternatively, consider designing a new Role Center if the activities for the activity button greatly vary from activities in other activity buttons.
Selecting multiple records in lists	Not available.	Not available.	Ctrl+A or Ctrl+Click on rows in a list using Microsoft Dynamics NAV Windows client.	Avoid scenarios requiring selecting multiple rows on a list. Also, try to minimize actions on lists.

Ribbon actions	Only Promoted actions are shown.	Only Promoted actions are shown.	On the Small Business Role Center.	Use the development environment to promote actions. Alternatively, configure the profile and add actions to the Home ribbon tab.
FactBoxes	Not shown on List pages or Worksheet pages.	Not shown on List pages or Worksheet pages.	Customer list on the Small Business Role Center.	Make sure the same information is visible on the corresponding card page of the given record.
Advanced filters	No column-specific filtering is available.	No column-specific filtering is available.	On the Customer list page.	Send data to Excel and do the complex filtering there.
Page search	Not available.	Not available.	On Microsoft Dynamics NAV Windows client or Microsoft Dynamics NAV Web client.	Design pages to expose the workflows needed by the user. For example via list places, tiles or actions.
Cue and action tiles	The number of tiles that are shown is based on the size of the screen, no possibility to scroll.	The number of tiles on the phone is in theory unlimited because you can scroll.	On most Role Center pages.	Design Role Center pages to avoid having important tiles at the area end. Assume you have no control over how many tiles are displayed and consider that only the first few tiles will be made visible on the tablet.
Fields in fast tabs	Fields in fast tabs on list pages are not shown. Only the repeater control is	Not available.		Design List pages to avoid having important columns on the

	shown in the content area of the page.			far right of the column list. Assume you have no control over how many columns are displayed and consider that only the first few columns will be made visible.
Links and Notes	Not available.	Not available.	On Sales Orders.	Similar to Factboxes, make sure the same information in the field group is visible on the corresponding card page of the given record.
Select from full list	Not available on lookups. Users are not able to run actions on a lookup page, and they cannot access the full set of records.	Not available on lookups. Users are not able to run actions on a lookup page, and they cannot access the full set of records.	On the Item Card when selecting the Base Units of Measure.	Make sure the appropriate columns are visible on the lookup. The user is still able to filter, scroll, and search through the lookup.
Search across list columns	Partly supported. Search will not include FlowFields.	Partly supported. Search will not include FlowFields.	On the Customer list page.	
Report Viewer and CurrReport.PREVIEW	There is no Preview functionality available on the Microsoft Dynamics NAV Tablet client. CurrReport.PREVIEW cannot be reliably used to identify whether a report was run as a draft or as a final printed document.	There is no Preview functionality available on the Microsoft Dynamics NAV Phone client. CurrReport.PREVIEW cannot be reliably used to identify whether a report was run as a draft or as a final printed document.	See example in report 280.	

Lookups	Available.	Available, with the difference that advanced and simple lookups behave similarly on the phone. The lookup will not bring up the card, show factboxes, or any field groups.	See examples on the Customer Card page.	
Matrix controls	Not available.	Not available.	See example in G/L Budget.	
File download	Available. Cannot download multiple files at the same time.	Available. Cannot download multiple files at the same time.	Trial Balance report in thePrint to Excel check box.	
Worksheet pages	Available.	Not available; an error message is displayed.	Sales Price Worksheet or Cash Flow Worksheet.	Run this type of page from the Microsoft Dynamics NAV Windows client, Microsoft Dynamics NAV Web client, or Microsoft Dynamics NAV Tablet client.
Lists	Available.	Available, with the difference that these are displayed in a brick layout with a number of differences and limitations. For an overview, see How to: Display Data as Bricks on Phone .	Customers or Sales Orders pages.	
Indentation in repeater controls	Available.	Not available. The repeater control will be rendered as a regular flat brick layout.	Chart of Accounts and Contacts List pages.	
Scope of actions	Available.	Available, but there are some behavioral differences regarding the Scope		

		Property . Also, see Defining Action Scope for Microsoft Dynamics NAV Pages .		
Use of camera and location	<p>Available in the Microsoft Dynamics NAV Universal App on devices with a camera and GPS capabilities.</p> <p>Note Not available on Microsoft Dynamics NAV Windows client or Microsoft Dynamics NAV Web client.</p>	<p>Available in the Microsoft Dynamics NAV Universal App on devices with a camera and GPS capabilities.</p> <p>Note Not available on Microsoft Dynamics NAV Windows client or Microsoft Dynamics NAV Web client.</p>	<p>On the Accounting Manager profile, under Incoming Documents</p>	



How to: Implement the Camera in C/AL

This example illustrates how you can add access to camera to a specific page from the development environment. Adding a camera option to the Item card, for example, lets you take a picture of a specific item and store it with the item. The example implements three actions; Take Picture, Take Picture High Quality, and Take Picture Low Quality on the Customer Card page, but does not include code that saves the picture to the database. For a Microsoft Dynamics NAV implementation of this, see Incoming Documents, for example on the Accounting Manager profile, when you use the Dynamics NAV app on a phone.

The camera access is only available on devices that run the Microsoft Dynamics NAV Universal App and have a camera. This means that camera access is not available from the Microsoft Dynamics NAV Windows client or from a browser.

With the following steps, you will create two variables; the CameraAvailable variable is a Boolean that checks whether the current device has a camera. The Camera variable is a DotNet type that gets instantiated by adding code to the OnOpenPage trigger. Then, you will add actions to the Customer Card page that lets the user start the camera and write the code that is run on these actions. And finally, you will add a new trigger Camera::PictureAvailable to handle the incoming picture.

To implement the camera in C/AL

1. In the development environment, on the Tools menu, choose Object Designer to open the Object Designer window.
2. In Object Designer, choose Pages, select the Customer Card (page 21) and choose the Design button.
3. From the Page Designer window, on the View menu, choose C/AL Globals.
4. Create the following two variables:

Variable name	Data Type	Sub Type
Camera	DotNet	Microsoft.Dynamics.Nav.Client.Capabilities.CameraProvider Important Choose the Microsoft.Dynamics.Nav.ClientExtensions dll on the Server tab, and then choose Microsoft.Dynamics.Nav.Client.Capabilities.CameraProvider. Make sure to set the properties RunOnClient and WithEvents to Yes.
CameraAvailable	Boolean	-

5. On the View menu, select C/AL Code and in the C/AL Editor locate the OnOpenPage trigger.
6. Instantiate the Camera variable by adding the following code to the OnOpenPage trigger

```
IF Camera.IsAvailable THEN  
BEGIN  
    Camera := Camera.Create;  
    CameraAvailable := TRUE;  
END;
```

7. Next, create the page actions. Choose the View menu, and then select Page Actions.

8. Locate the ActionGroup named Customer and create three actions; TakePicture, TakePictureHigh, and TakePictureLow all of them with the following properties set as shown in the following table.

Property	Value
Name	TakePicture
Visible	CameraAvailable
Promoted	Yes
PromotedCategory	Process
PromotedIsBig	Yes
Image	Camera

9. Next, you will add the code that is executed on the actions. On the TakePicture OnAction trigger, insert the following line of code:

```
Camera.RequestPictureAsync;
```

10. On the TakePictureHigh OnAction trigger, insert the following lines of code:

```
CameraOptions := CameraOptions.CameraOptions();
CameraOptions.Quality := 100;
Camera.RequestPictureAsync(CameraOptions);
```

11. On the TakePictureLow OnAction trigger, insert the following lines of code:

```
CameraOptions := CameraOptions.CameraOptions();
CameraOptions.Quality := 10;
Camera.RequestPictureAsync(CameraOptions);
```

12. You now need to declare the local variable CameraOptions for the TakePictureHigh and TakePictureLow triggers. From the Page Designer window, on the View menu, choose C/AL Locals.

13. Create the following variable:

Variable name	Data Type	Sub Type
CameraOptions	DotNet	Microsoft.Dynamics.Nav.Client.Capabilities.CameraOptions Important Choose the Microsoft.Dynamics.Nav.ClientExtensions dll on the Server tab, and then choose Microsoft.Dynamics.Nav.Client.Capabilities.CameraOptions.

14. You must now add code to handle the picture for when the camera has captured the picture and the picture has been uploaded to the Microsoft Dynamics NAV Server. In the C/AL Editor, on the PictureAvailable trigger, add code so that the PictureAvailable trigger looks like this.


```
Camera::PictureAvailable(PictureName : Text;PictureFilePath : Text)
    IncomingFile.OPEN(PictureFilePath);
    MESSAGE('Picture size: %1', IncomingFile.LEN);
    IncomingFile.CLOSE;
    FILE.ERASE(PictureFilePath);
```

The PictureName contains the name of the file including its extension on the device. The naming scheme depends on the device platform. The PictureFilePath contains the path to the picture in a temporary folder on the server for the current user.

It is important to clean up by using the FILE.ERASE command to avoid accumulating image files.

- Now, you need to declare the local variable IncomingFile used in the PictureAvailable trigger. From the Page Designer window, on the View menu, choose C/AL Locals.
- Create the following variable:

Variable name	Data Type
IncomingFile	File

- Close the C/AL Editor and then save and compile the page

You can now test the modified Customer Card page in the Microsoft Dynamics NAV Universal App from either a tablet or a phone with a camera.

How to: Implement Location in C/AL

This example illustrates how you can retrieve location information. The example implements a **GetLocation** action on the Customer Card (page 21) that returns the GPS coordinates of the current customer's address. It does not save this information to the database. Scenarios in which this functionality could be useful would be displaying a map that shows where your customer is located based on the GPS coordinates. Or, functionality to plan the next round of customer visits based on the addresses of your customers.

The location information is only available on devices that run the Microsoft Dynamics NAV Universal App and have GPS capabilities. This means that location information is not available from the Microsoft Dynamics NAV Windows client or from a browser.

To implement location in C/AL

1. In the development environment, on the Tools menu, choose Object Designer to open the Object Designer window.
2. In Object Designer, choose Pages, select the Customer Card (page 21) and then choose the Design button.
3. From the Page Designer window, on the View menu, choose C/AL Globals.
4. Create the following variable:

Variable name	Data Type	Sub Type
Location	DotNet	Microsoft.Dynamics.Nav.Client.Capabilities.LocationProvider Important Choose the Microsoft.Dynamics.Nav.ClientExtensions.dll on the Server tab, and then choose Microsoft.Dynamics.Nav.Client.Capabilities.LocationProvider. Make sure to set the properties RunOnClient and WithEvents to Yes.
LocationAvailable	Boolean	-

5. On the View menu, select C/AL Code and in the C/AL Editor locate the OnOpenPage trigger.
6. Instantiate the Location variable by adding the following code to the OnOpenPage trigger.

```
IF Location.IsAvailable THEN
BEGIN
    Location := Location.Create;
    LocationAvailable := TRUE;
END;
```

7. Next, create the page action. Choose the View menu, and then select Page Actions.
8. Locate the ActionGroup named Customer and create a new action; GetLocation with the following properties.

Property	Value
Name	GetLocation
Visible	LocationAvailable
Promoted	Yes
PromotedCategory	Process
PromotedIsBig	Yes

9. Now, in the C/AL Editor, on the GetLocation – OnAction trigger, insert the following line of code.

```
Location.RequestLocationAsync;
```

10. While still in the C/AL Editor, on the LocationChanged trigger add the following code to handle the GPS coordinates. LocationChanged is called when the device has obtained a status.

```
Location::LocationChanged(Location : DotNet "Microsoft.Dynamics.Nav.Client.Capabilities.Location")
IF(Location.Status = 0) THEN
MESSAGE('Your position: %1 %2', Location.Coordinate.Latitude, Location.Coordinate.Longitude)
ELSE
MESSAGE('Position not available');
```

Location.Status can be 0 = Available, 1 = NoData (no data could be obtained), 2 = TimedOut (location information not obtained in due time), or 3 = NotAvailable (for example user denied app access to location).

11. Close the C/AL Editor, and then save and compile the page.
12. You can now test the modified Customer Card page in the Microsoft Dynamics NAV Universal App from either a tablet or a phone with GPS capabilities.