
POWERED BY DUG
DYNAMICSCON
LIVE
MAY 2024
live.dynamicscon.com

Give the User a Chance

Just because it works, doesn't mean it works ...

Henrik Helgesen | Helgesen Consulting, Inc.

Henrik Helgesen – Aka TheDoubleH

- Originally from Denmark.
- Currently live in Los Angeles, California
- Started with Navigator 3.04 in 1995
- Have worked in all aspects from before life, to death – all in NAV / BC
- Stickler for rules – Previously known as the GUI Police
- Long Time Listener, First time Caller
- Hockey Fan (Los Angeles Kings)
- Magic Fan (Magic Castle Member)

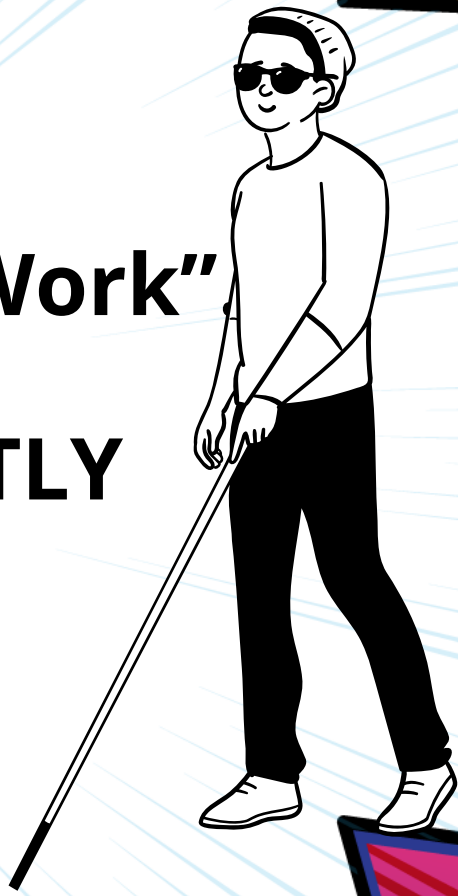
Yes – I am a Dinosaur

Objective

Share some tips on what options you as a developer have to help guide the user in their day-to-day work

As Developers we tend to “make it Work”

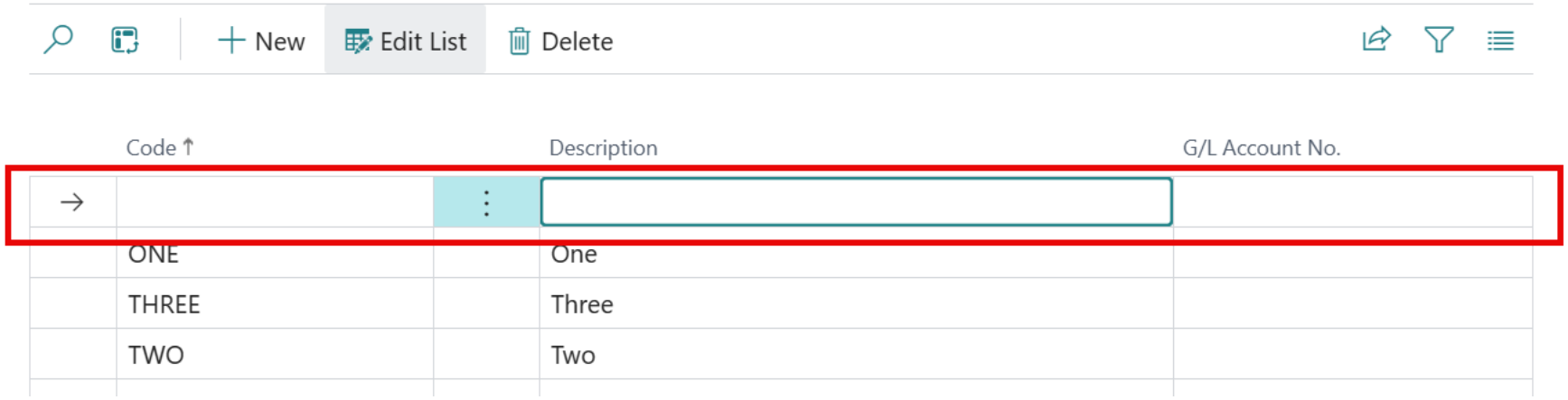
… but only if the user does EXACTLY what we expect them to do …



What can we do?

Prevent Bad Data

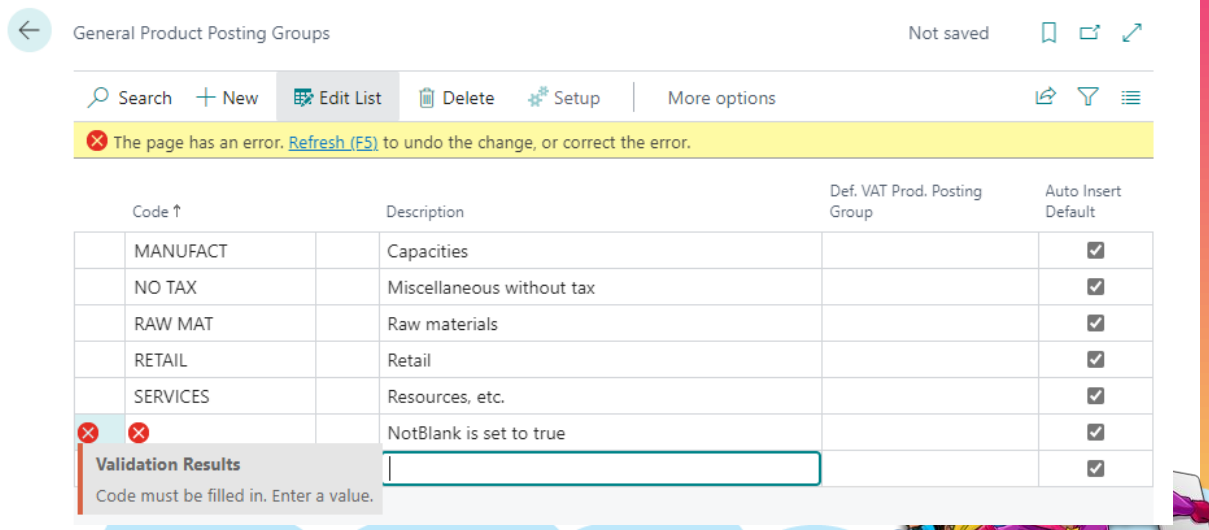
Prevent <Blank> Records



Prevent <Blank> Records

- As a Developer, You can ensure that the Primary Key fields is populated.
- Base App: Table 251 – Gen. Product Posting Group
- The `Code` field has NotBlank = true

```
1  table 251 "Gen. Product Posting Group"
2  {
3      Caption = 'Gen. Product Posting Group';
4      DataCaptionFields = "Code", Description;
5      LookupPageID = "Gen. Product Posting Groups";
6
7      fields
8      {
9          field(1; "Code"; Code[20])
10         {
11             Caption = 'Code';
12             NotBlank = true;
13         }
14         field(2; Description; Text[100])
15         {
16             Caption = 'Description';
17         }
18         field(3; "Def. VAT Prod. Posting Group"; Code[20])
19         {
20             Caption = 'Def. VAT Prod. Posting Group';
21             TableRelation = "VAT Product Posting Group";
```



Prevent <Blank> Records

Property: **`NotBlank`**

Sets a value that specifies whether users must enter a value in the selected field or text box.

Only “valid” for Primary Key Fields.

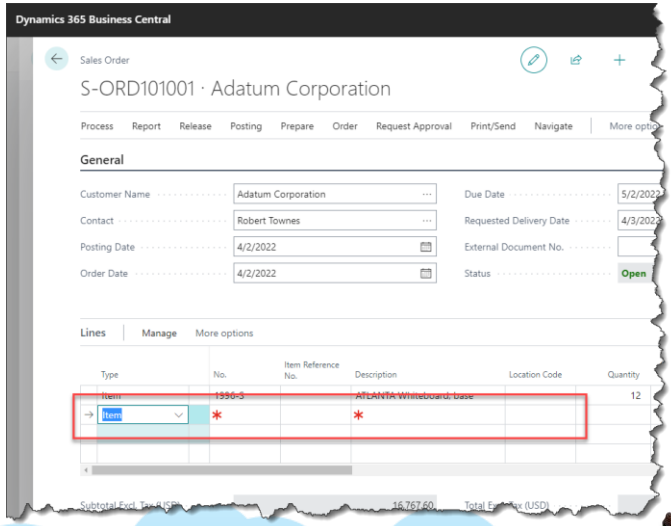
Only Primary Key Fields are enforced on OnInsert, but other fields are enforced – if you happen to enter / clear a value.

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/properties/devenv-

Indicate Mandatory fields for processing

On pages, You can set ShowMandatory to true. This will place a red asterisk in the text box, giving the user a visual indicator that a value is required.

```
54              field("No."; "No.")
55              {
56                  ApplicationArea = Basic, Suite;
57                  ShowMandatory = Type <> Type::" ";
58                  ToolTip = 'Specifies the number of a general ledger account, item, resource, additional c
59
60                  trigger OnValidate()
61                  begin
62                      NoOnAfterValidate();
63                      UpdateEditableOnRow();
64                      ShowShortcutDimCode(ShortcutDimCode);
65
```



POWERED BY DUG
DYNAMICSCON LIVE
MAY 2024

Indicate Mandatory fields for processing

Property: **`ShowMandatory`**

Sets a value that specifies whether users must enter a value in the selected field or text box. The field is marked on the page with a red asterisk and does not enforce any validation. Once the field is filled, the red asterisk disappears. The ShowMandatory property only controls the UI and overrides any asterisk marking of the NotBlank Property.

Also, be aware that while it is possible to use an expression for the ShowMandatory property, the property cannot validate an AL function.

For an example, see the Purchase Invoice page where the Vendor Invoice No field is controlled by the VendorInvoiceNoMandatory expression. VendorInvoiceNoMandatory is a variable that takes the value from the Ext. Doc. No. Mandatory field on the Purchase page and the Payables page.

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/properties/devenv-showmandatory-property

Indicate Mandatory fields for processing

Property: **`ShowMandatory`**

Sets a value that specifies whether users must enter a value in the selected field or text box. The field is marked on the page with a red asterisk and does not enforce any validation. Once the field is filled, the red asterisk disappears. The ShowMandatory property only controls the UI and overrides any asterisk marking of the NotBlank Property.

Also, be aware that while it is possible to use an expression for the ShowMandatory property, the property cannot validate an AL function.

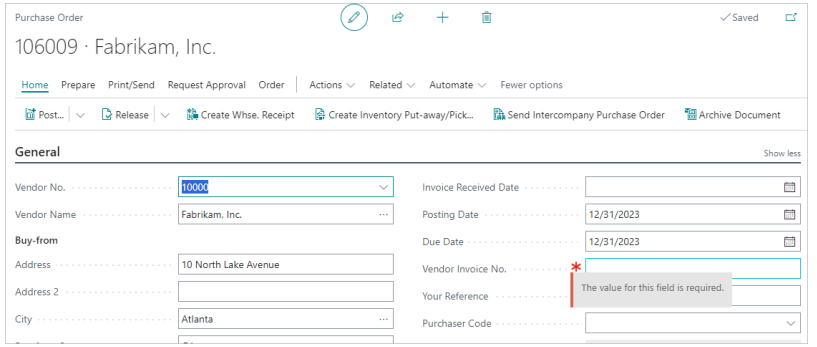
For an example, see the Purchase Invoice page where the Vendor Invoice No field is controlled by the VendorInvoiceNoMandatory expression. VendorInvoiceNoMandatory is a variable that takes the value from the Ext. Doc. No. Mandatory field on the Purchase page and the Payables page.

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/properties/devenv-showmandatory-property

Steve Endow

@steveendow

Why is the Vendor Invoice No field marked as required for a brand new Purchase Order? 😕 #MSDyn365BC



12:09 PM · Apr 14, 2024

NotBlank

≠

ShowMandatory

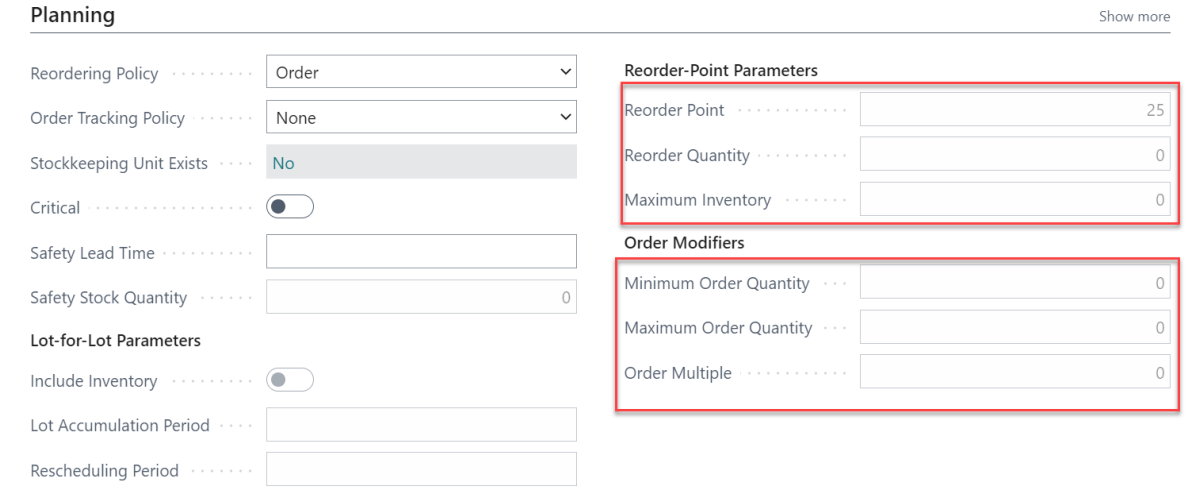
ShowMandatory

Is not

Enforced

Enabled (true/false)

```
field("Reorder Quantity"; "Reorder Quantity")
{
    ApplicationArea = Planning;
    Enabled = ReorderQtyEnable;
    ToolTip = 'Specifies a standard lot size quantity to be used for all order proposa
}
```



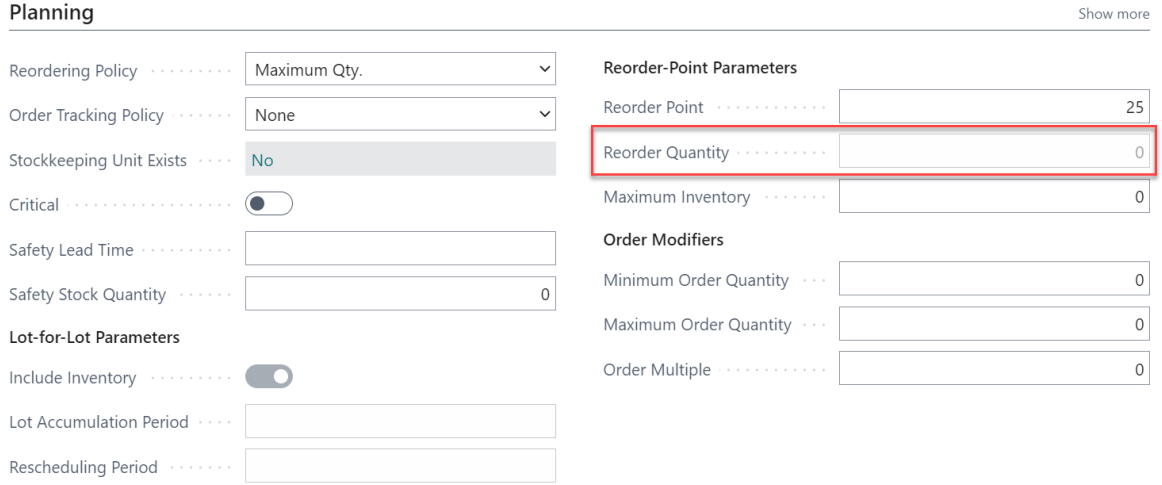
Manipulating the Enabled Property keeps the field on the page but enables control whether field can be edited or not.

This can be good when another field controls what information is needed.

The Item Card is a good example of it's use.

Enabled (true/false)

```
field("Reorder Quantity"; "Reorder Quantity")
{
    ApplicationArea = Planning;
    Enabled = ReorderQtyEnable;
    ToolTip = 'Specifies a standard lot size quantity to be used for all order proposa
}
```



Manipulating the Enabled Property keeps the field on the page but enables control whether field can be edited or not.

This can be good when another field controls what information is needed.

The Item Card is a good example of it's use.

Visible (true/false)

Visible should only be set OnOpen. Otherwise, the page can be jumbled and confusing.

Works well for Card pages for tables that require different fields populated based on an already known value. i.e.:

- Credit Card Apps
- Transaction Type
- Card Issuer
- Gateway
- Shipping Solutions
- Carrier
- Domestic vs International

```
131     trigger OnOpenPage()
132     begin
133         IsCountyVisible := FormatAddress.UseCounty("Country/Region Code");
134     end;
135
136 var
```

```
36                  group(Control5)
37                  {
38                      ShowCaption = false;
39                      Visible = IsCountyVisible;
40                      field(County; County)
41                      {
42                          ApplicationArea = Basic, Suite;
43                      }
```

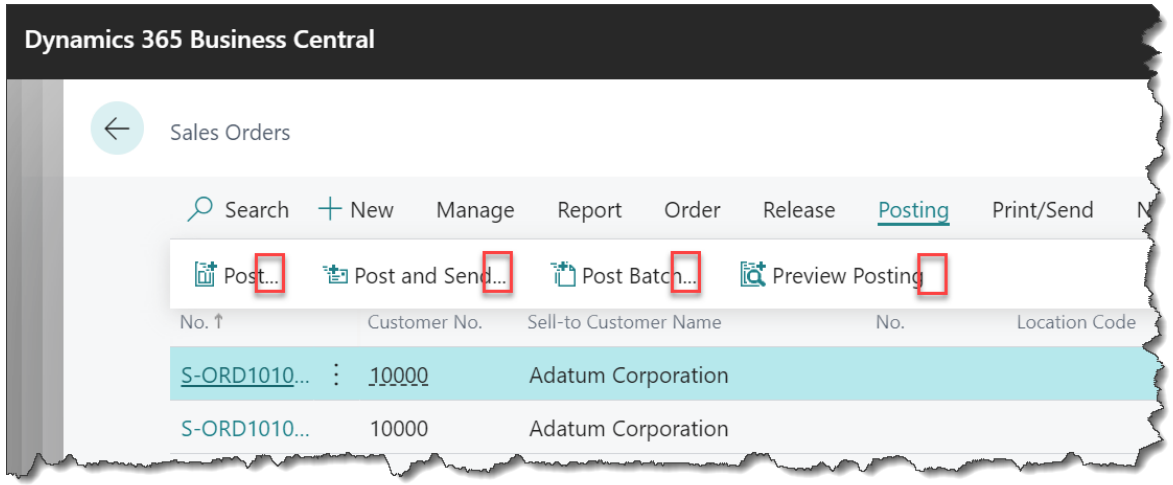
Indicators

Ellipsis (Three dots)

Official Documentation says: “Sets a value that specifies whether an ellipsis (…) is appended to the caption on a command button or menu item.

An ellipsis tells the user that other choices will appear if the command button or menu item are selected.”

To me – It’s an indicator that ‘nothing bad’ will happen if You press this action.



Show something is happening

Unless You, as a developer, do anything, Users are kept in the dark about what is happening.

All they are shown is this:

Working on it ...

Show something is happening

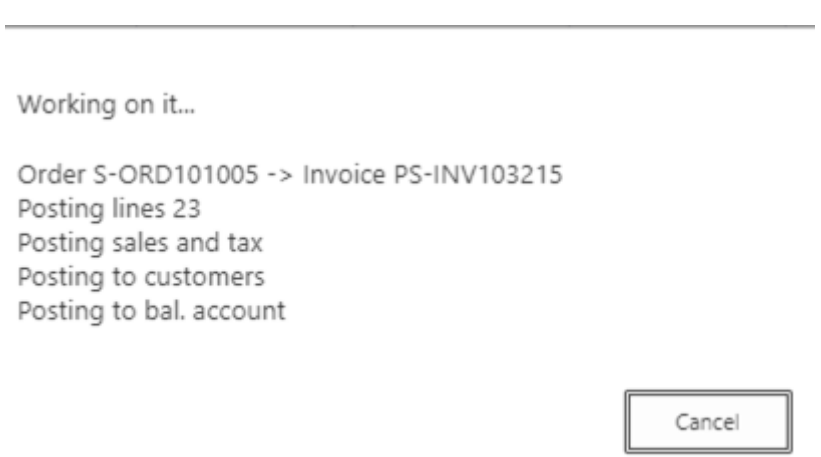
When processing data, that could take longer than one second, I always show the user, that something is happening.

Using the Dialog variable type, allows for ‘communication’ to the user – and more importantly – gives the user a ‘Cancel’ button.

Window.Update takes a lot of resources, so You will want to be cautious, and not update too rapidly.

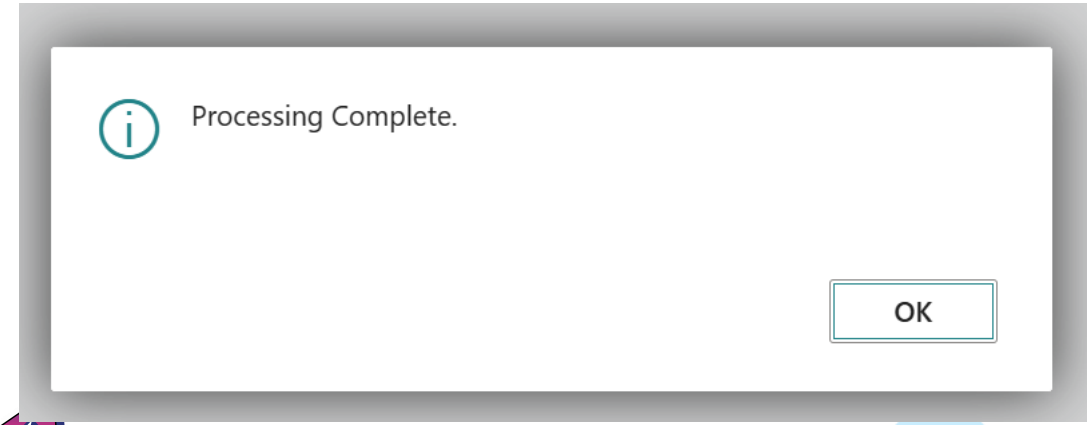
You really shouldn’t need to update more often than ever second.

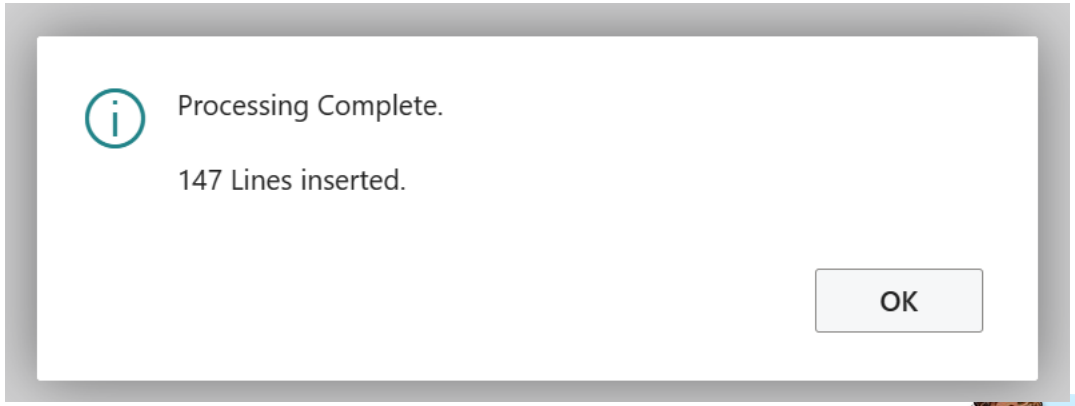
```
if TempSalesLineGlobal.FindSet() then
    repeat
        ErrorMessageMgt.PushContext(ErrorContextElementPostLine, TempSalesLineGlobal.RecordId, 0, PostDocumentLinesMsg);
        ItemJnlRollRndg := false;
        LineCount := LineCount + 1;
        if not HideProgressWindow then
            Window.Update(2, LineCount);

        PostSalesLine(
          SalesHeader, TempSalesLineGlobal, EverythingInvoiced, TempVATAmountLine, TempVATAmountLineRemainder,
          TempItemLedgEntryNotInvoiced, HasATOShippedNotInvoiced, TempDropShptPostBuffer, ICGenJnlLineNo,
          TempServiceItem2, TempServiceItemComp2);

        if RoundingLineInserted then
            LastLineRetrieved := true
        else begin
            BiggestLineNo := MAX(BiggestLineNo, TempSalesLineGlobal."Line No.");
            LastLineRetrieved := TempSalesLineGlobal.Next() = 0;
            if LastLineRetrieved and SalesSetup."Invoice Rounding" then
                InvoiceRounding(SalesHeader, TempSalesLineGlobal, false, BiggestLineNo);
        end;
        OnRunOnBeforePostSalesLineEndLoop(SalesHeader, TempSalesLineGlobal, LastLineRetrieved, SalesInvHeader, SalesCrMemoHeader, Rec, xSalesLine
        ErrorMessageMgt.PopContext(ErrorContextElementPostLine);
```



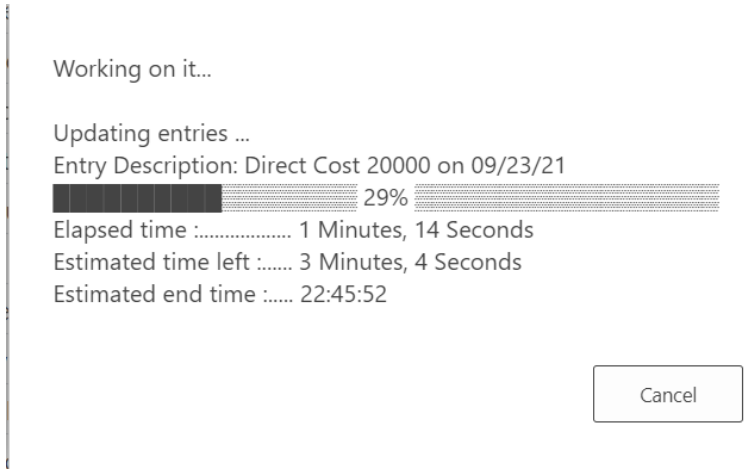
Show something is happening

… and show something when you are done.





My gift to you ...




```
procedure UpdateWindow(Counter: Integer; NoOfRecords: Integer);
var
    ProgressBar: Codeunit "Progress Bar TBHLG";
    EndTime: DateTime;
    CurrDuration: Duration;
    EstimatedDuration: Duration;

begin
    if CurrentDateTime < LastUpdate + 1000 then
        exit;

    if Counter = 0 then
        exit;

    Window.Update(20, ProgressBar.ProgressBar(Counter, NoOfRecords));
    LastUpdate := CurrentDateTime;
```

```
var
    GLEntry: Record "G/L Entry";
    DialogHelperTBHLG: Codeunit "Dialog Helper TBHLG";
    NoOfRecords: Integer;
    OnRecordNo: Integer;
begin
    DialogHelperTBHLG.OpenWindow('Updating entries ...\\Entry Description: #1################', true);
    NoOfRecords := GLEntry.Count();
    GLEntry.FindSet();
    repeat
        OnRecordNo += 1;
        DialogHelperTBHLG.UpdateWindow(1, GLEntry.Description, OnRecordNo, NoOfRecords);
        sleep(50)
    until GLEntry.next() = 0;
end;
```

In my BCALToolbox (Open Source https://go.thedoubleh.dev/toolbox) I have created function, that make it easy.

These functions makes sure I only update once every second.

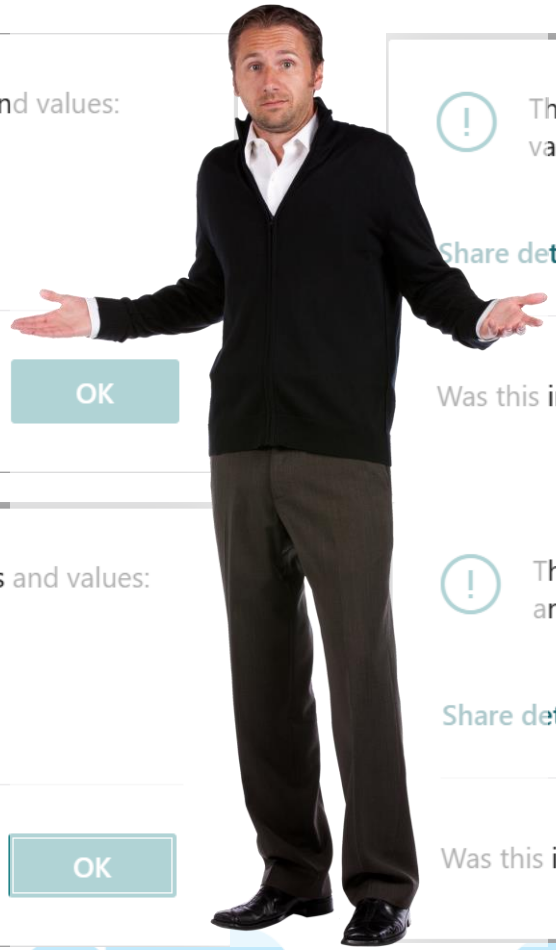
Meaningful Errors

Give the user a reference

The Customer does not exist. Identification fields and values: No.=''

Share details

Was this information helpful? Yes No OK



The Payment Terms does not exist. Identification fields and values: Code=''

Share details

Was this information helpful? Yes No OK

The G/L Account does not exist. Identification fields and values: No.=''

Share details

Was this information helpful? Yes No OK

The Customer Posting Group does not exist. Identification fields and values: Code=''

Share details

Was this information helpful? Yes No OK

Give the user a reference

If you do not give the user a refence, how will they know what is wrong?

Using a testfield will make sure a value is there (or will make sure that the value is a specific value)

As a child, I was always taught:

“Always use a testfield, if you are dependent of the value”

Give the user a reference

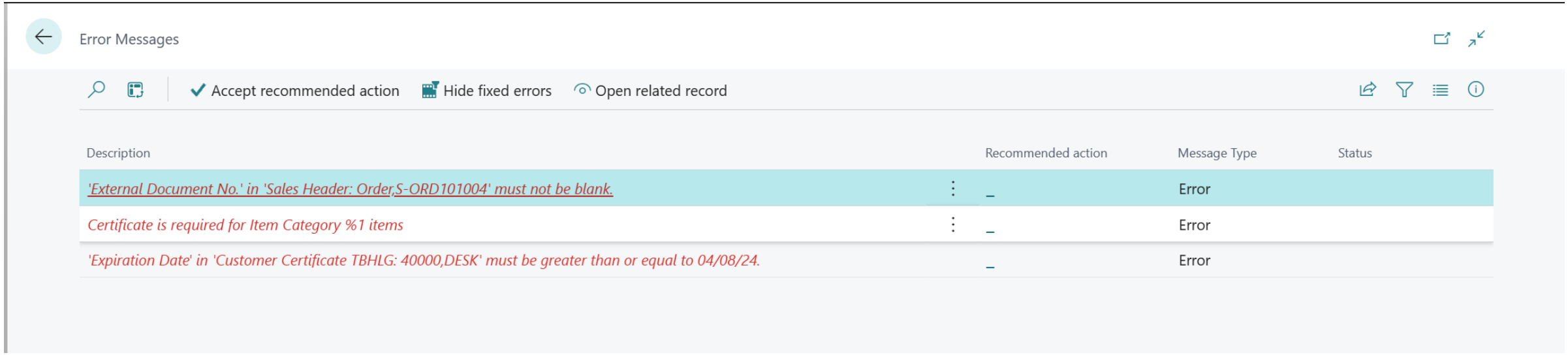
Method: **`testfield`**

Tests that the content of the field is not zero or blank (empty string). If it is, an error message is displayed.

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/methods-auto/fieldref/fieldref-testfield--method

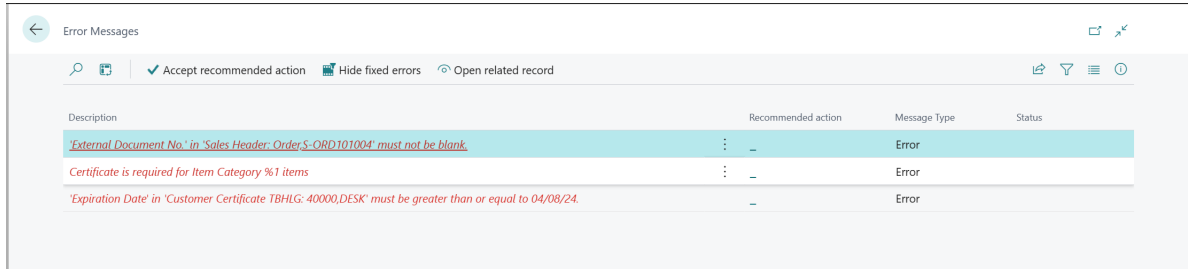

BUT ...
I use the testfield as a backup

Show multiple errors



Show multiple errors

Use the Error Message Table to collect errors, and present at the end.



Show multiple errors

The Error Message table has several procedures (methods) you can call:

LogIfEmpty(RecRelatedVariant: Variant, FieldNumber: Integer, MessageType: Option): Integer

LogIfLessThan(RecRelatedVariant: Variant, FieldNumber: Integer, MessageType: Option, UpperBound: Variant): Integer

LogDetailedMessage(RecRelatedVariant: Variant, FieldNumber: Integer, MessageType: Option, NewDescription: Text, AdditionalInformation: Text[250], SupportUrl: Text[250]): Integer

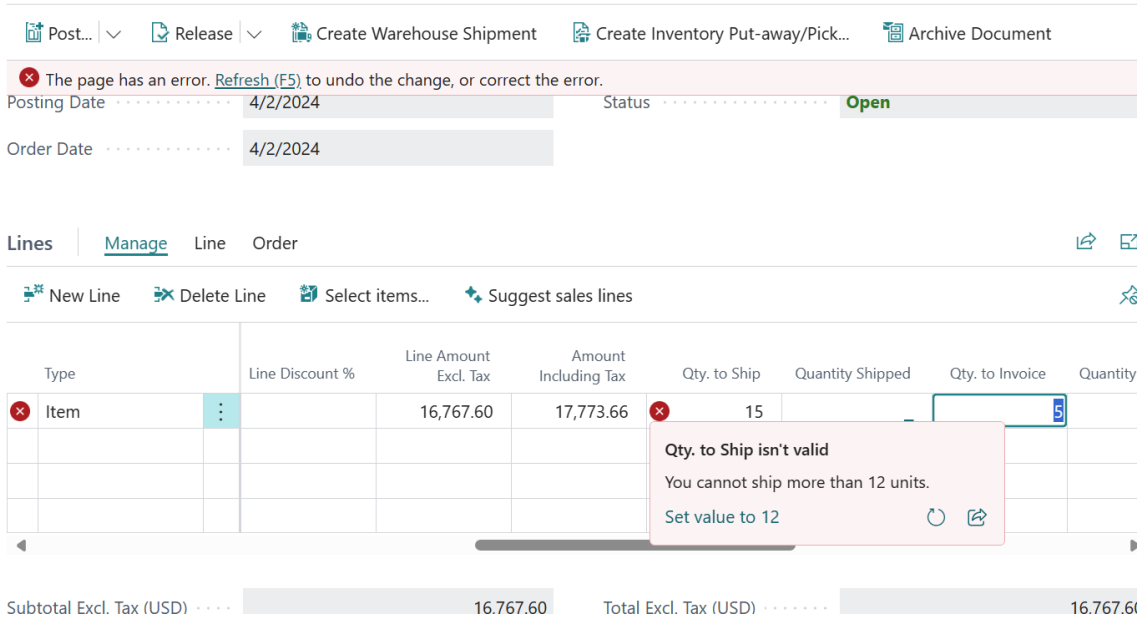
Actionable Error Message

While there, sadly, isn't any example of this in the Base app, I wholeheartedly believe in the use of the Error Message Table (Error Management).

When I release a 'Document', I will run through the header and lines, and test for 'mandatory' fields (Those have been flagged as ShowMandatory).

I will also test if the affected record is 'blocked' or otherwise invalid.

Actionable Error Message

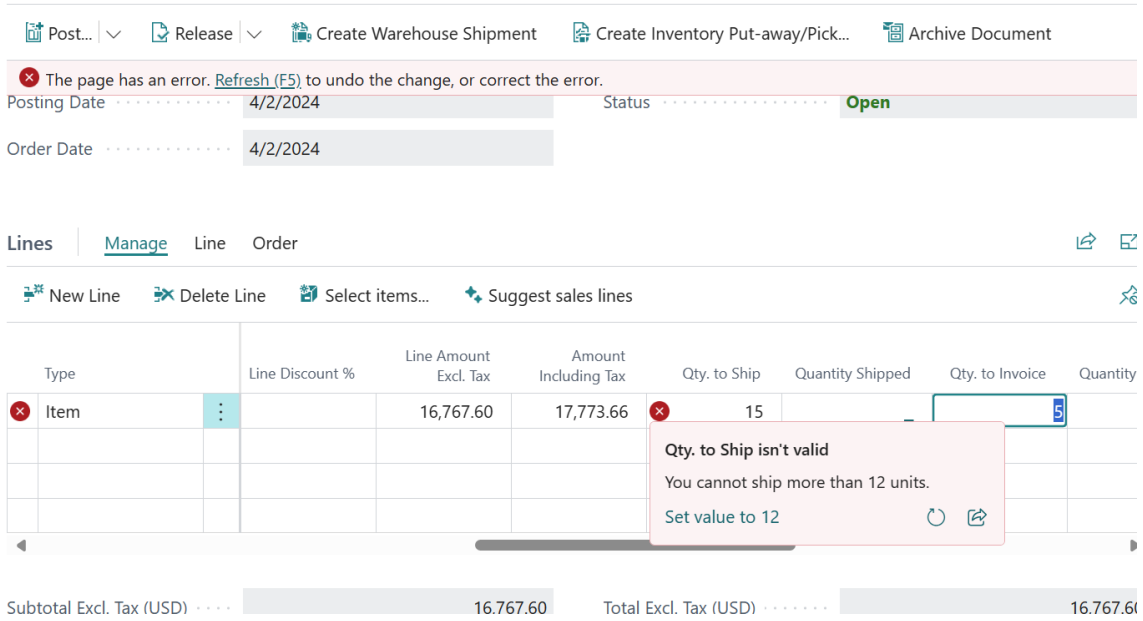


Use the Actionable Error Message

Using the Error Message Management Codeunit, you can call a codeunit to “fix it” for the user.

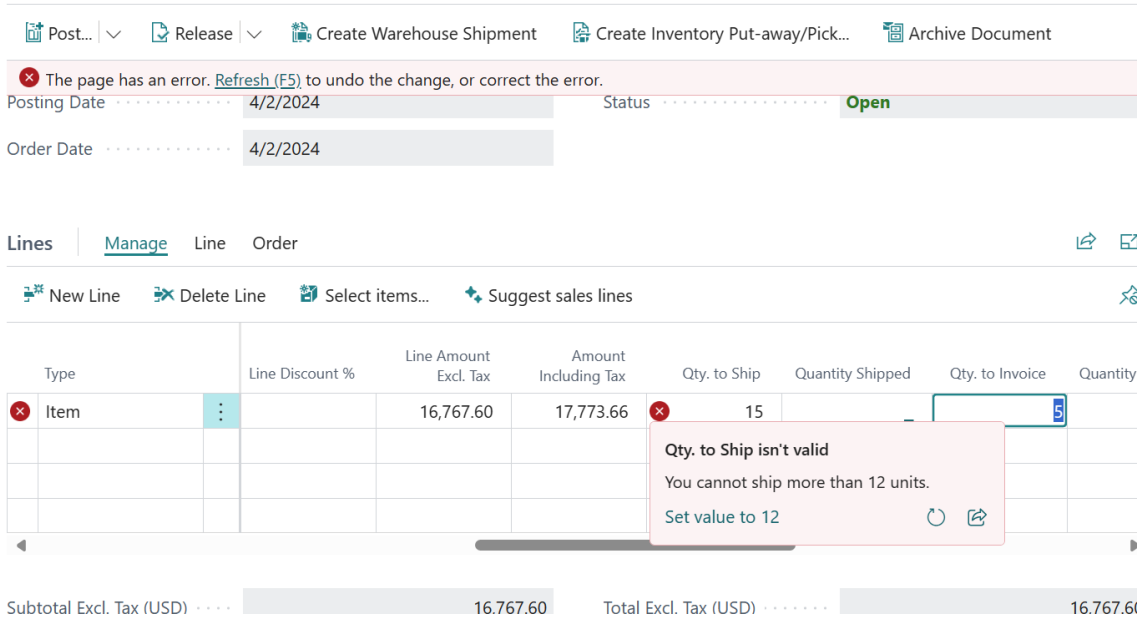
SalesLine.”Qty. to Ship” validation is a great example

Actionable Error Message



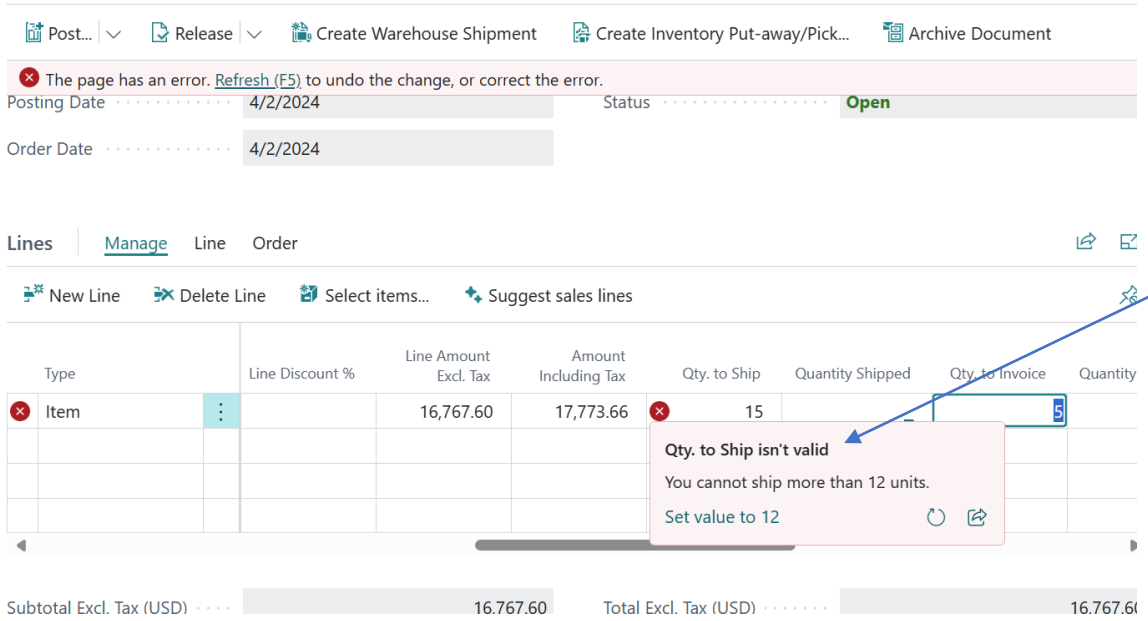

```
local procedure CannotShipErrorInfo(): ErrorInfo
    var
        ErrorMesageManagement: Codeunit "Error Message Management";
    begin
        exit(ErrorMesageManagement.BuildActionableErrorInfo(
            QtyShipNotValidTitleLbl,
            StrSubstNo(Text007, "Outstanding Quantity"),
            Rec.RecordId,
            StrSubstNo(QtyShipActionLbl, "Outstanding Quantity"),
            Codeunit::"Sales Line-Reserve",
            'SetSaleShipQty',
            StrSubstNo(QtyShipActionDescriptionLbl,
              Rec.FieldCaption("Qty. to Ship"), Rec.Quantity)));
    end;
```

Actionable Error Message



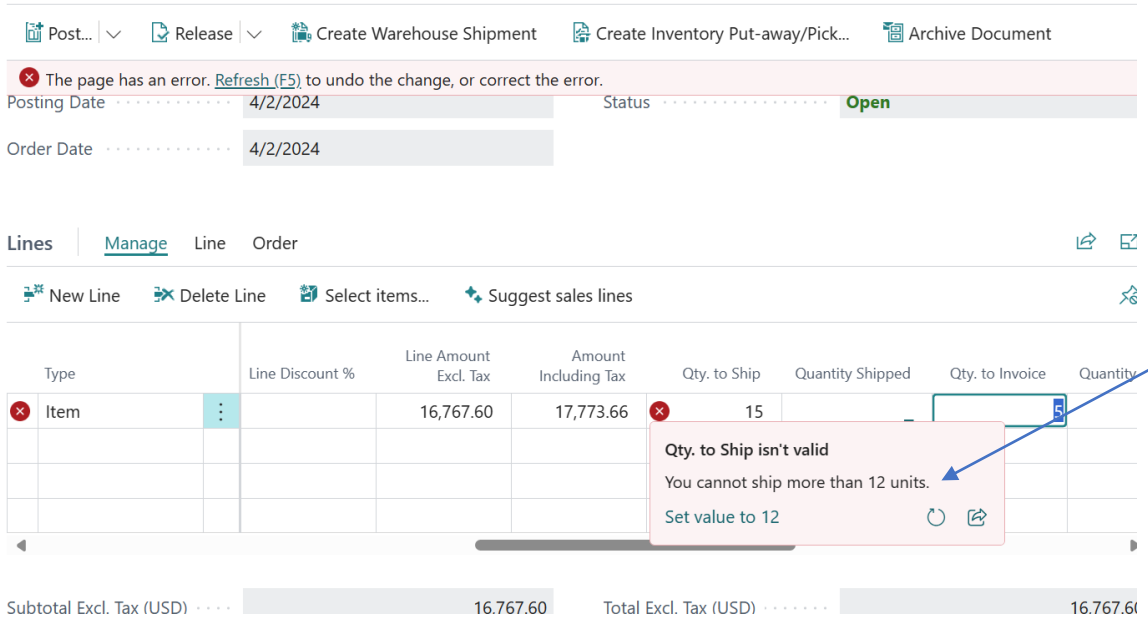

```
local procedure CannotShipErrorInfo(): ErrorInfo
    var
        ErrorMesageManagement: Codeunit "Error Message Management";
    begin
        exit(ErrorMesageManagement.BuildActionableErrorInfo(
            QtyShipNotValidTitleLbl,
            StrSubstNo(Text007, "Outstanding Quantity"),
            Rec.RecordId,
            StrSubstNo(QtyShipActionLbl, "Outstanding Quantity"),
            Codeunit::"Sales Line-Reserve",
            'SetSaleShipQty',
            StrSubstNo(QtyShipActionDescriptionLbl,
              Rec.FieldCaption("Qty. to Ship"), Rec.Quantity)));
    end;
```

Actionable Error Message



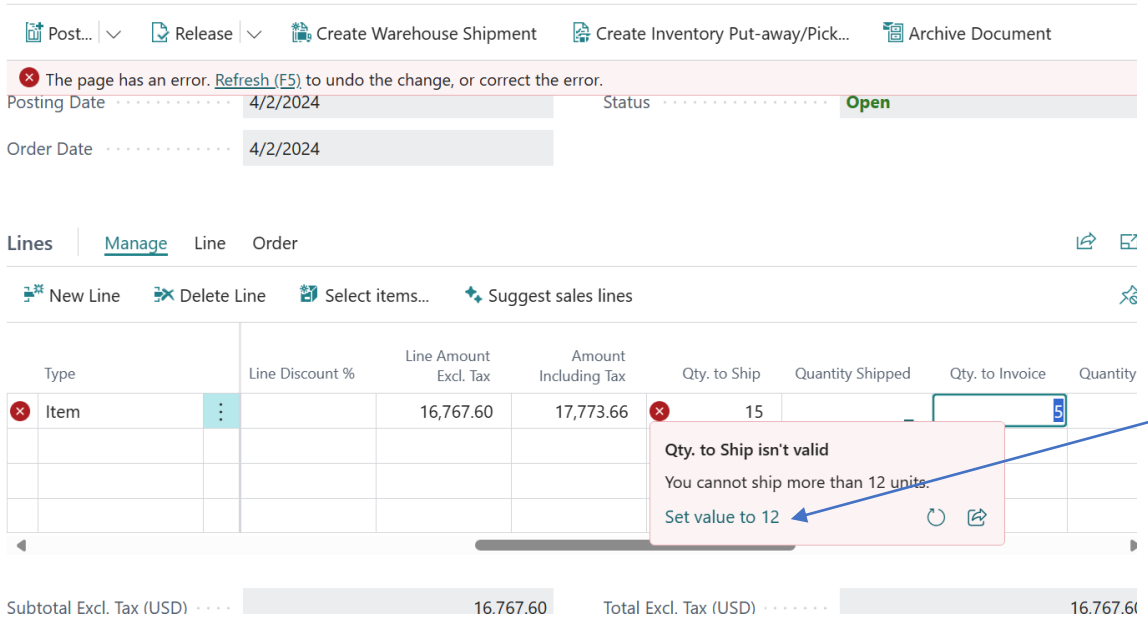

```
local procedure CannotShipErrorInfo(): ErrorInfo
    var
        ErrorMesageManagement: Codeunit "Error Message Management";
    begin
        exit(ErrorMesageManagement.BuildActionableErrorInfo(
            QtyShipNotValidTitleLbl,
            StrSubstNo(Text007, "Outstanding Quantity"),
            Rec.RecordId,
            StrSubstNo(QtyShipActionLbl, "Outstanding Quantity"),
            Codeunit::"Sales Line-Reserve",
            'SetSaleShipQty',
            StrSubstNo(QtyShipActionDescriptionLbl,
              Rec.FieldCaption("Qty. to Ship"), Rec.Quantity)));
    end;
```

Actionable Error Message



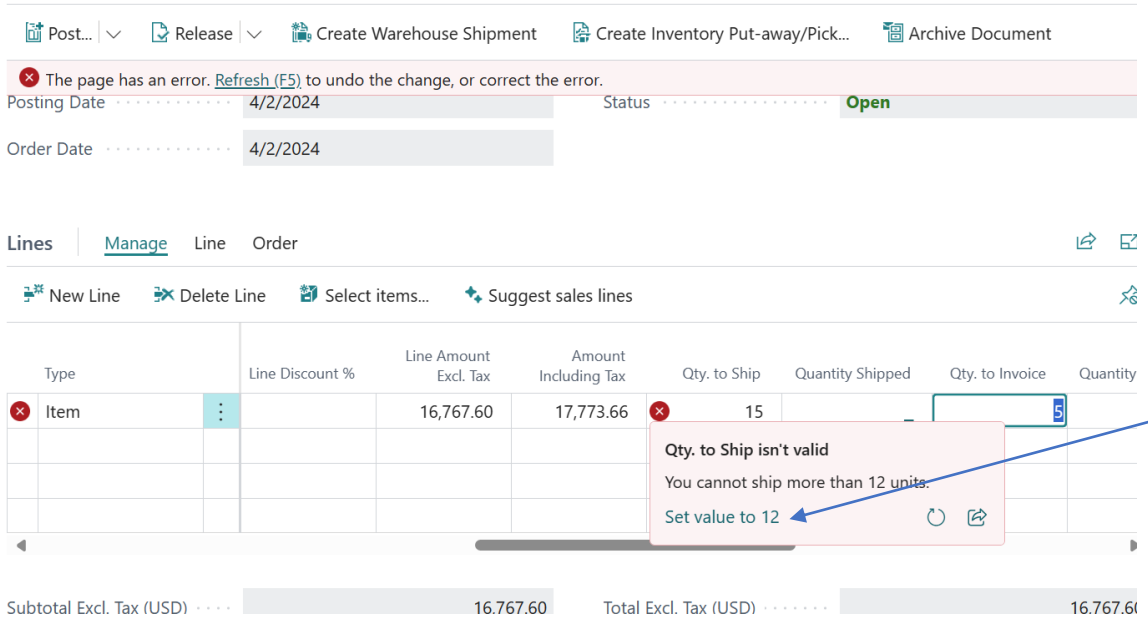

```
local procedure CannotShipErrorInfo(): ErrorInfo
    var
        ErrorMesageManagement: Codeunit "Error Message Management";
    begin
        exit(ErrorMesageManagement.BuildActionableErrorInfo(
            QtyShipNotValidTitleLbl,
            StrSubstNo(Text007, "Outstanding Quantity"),
            Rec.RecordId,
            StrSubstNo(QtyShipActionLbl, "Outstanding Quantity"),
            Codeunit::"Sales Line-Reserve",
            'SetSaleShipQty',
            StrSubstNo(QtyShipActionDescriptionLbl,
              Rec.FieldCaption("Qty. to Ship"), Rec.Quantity)));
    end;
```

Actionable Error Message



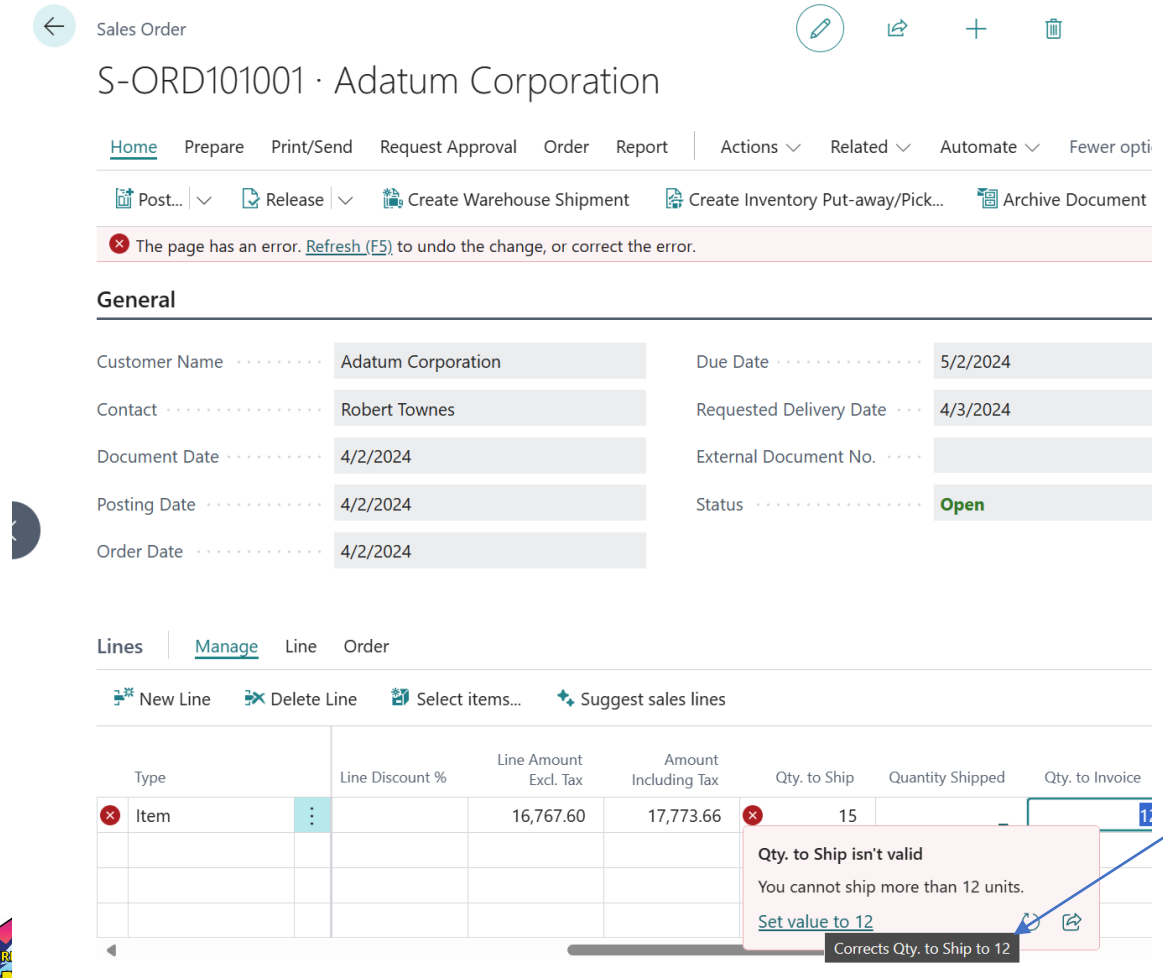

```
local procedure CannotShipErrorInfo(): ErrorInfo
    var
        ErrorMesageManagement: Codeunit "Error Message Management";
    begin
        exit(ErrorMesageManagement.BuildActionableErrorInfo(
            QtyShipNotValidTitleLbl,
            StrSubstNo(Text007, "Outstanding Quantity"),
            Rec.RecordId,
            StrSubstNo(QtyShipActionLbl, "Outstanding Quantity"),
            Codeunit::"Sales Line-Reserve",
            'SetSaleShipQty',
            StrSubstNo(QtyShipActionDescriptionLbl,
              Rec.FieldCaption("Qty. to Ship"), Rec.Quantity)));
    end;
```

Actionable Error Message




```
local procedure CannotShipErrorInfo(): ErrorInfo
    var
        ErrorMesageManagement: Codeunit "Error Message Management";
    begin
        exit(ErrorMesageManagement.BuildActionableErrorInfo(
            QtyShipNotValidTitleLbl,
            StrSubstNo(Text007, "Outstanding Quantity"),
            Rec.RecordId,
            StrSubstNo(QtyShipActionLbl, "Outstanding Quantity"),
            Codeunit::"Sales Line-Reserve",
            'SetSaleShipQty',
            StrSubstNo(QtyShipActionDescriptionLbl,
              Rec.FieldCaption("Qty. to Ship"), Rec.Quantity)));
    end;
```

Actionable Error Message




```
local procedure CannotShipErrorInfo(): ErrorInfo
    var
        ErrorMesageManagement: Codeunit "Error Message Management";
    begin
        exit(ErrorMesageManagement.BuildActionableErrorInfo(
            QtyShipNotValidTitleLbl,
            StrSubstNo(Text007, "Outstanding Quantity"),
            Rec.RecordId,
            StrSubstNo(QtyShipActionLbl, "Outstanding Quantity"),
            Codeunit::"Sales Line-Reserve",
            'SetSaleShipQty',
            StrSubstNo(QtyShipActionDescriptionLbl,
              Rec.FieldCaption("Qty. to Ship"), Rec.Quantity)));
    end;
```

BuildActionableErrorInfo

```
procedure BuildActionableErrorInfo(ErrorTitle: Text; ErrorMessage: Text; RecId: RecordId; ActionMessage: Text;
ActionCodeunuitId: Integer; ActionName: Text; ActionDescription: Text): ErrorInfo
    var
        ReturnErrorInfo: ErrorInfo;
    begin
        ReturnErrorInfo.Title := ErrorTitle;
        ReturnErrorInfo.Message := ErrorMessage;
        ReturnErrorInfo.RecordId := RecId;
        if ActionDescription <> '' then
            ReturnErrorInfo.AddAction(ActionMessage, ActionCodeunuitId, ActionName, ActionDescription)
        else
            ReturnErrorInfo.AddAction(ActionMessage, ActionCodeunuitId, ActionName);
        exit(ReturnErrorInfo);
    end;
```

BUT WAIT.

There’s more

BuildActionableErrorInfo

```
procedure BuildActionableErrorInfo(ErrorTitle: Text; ErrorMessage: Text; RecId: RecordId; ActionMessage: Text;
ActionCodeunuitId: Integer; ActionName: Text; ActionDescription: Text): ErrorInfo
    var
        ReturnErrorInfo: ErrorInfo;
    begin
        ReturnErrorInfo.Title := ErrorTitle;
        ReturnErrorInfo.Message := ErrorMessage;
        ReturnErrorInfo.RecordId := RecId;
        if ActionDescription <> '' then
            ReturnErrorInfo.AddAction(ActionMessage, ActionCodeunuitId, ActionName, ActionDescription)
        else
            ReturnErrorInfo.AddAction(ActionMessage, ActionCodeunuitId, ActionName);
        exit(ReturnErrorInfo);
    end;
```

Add a Second Action!

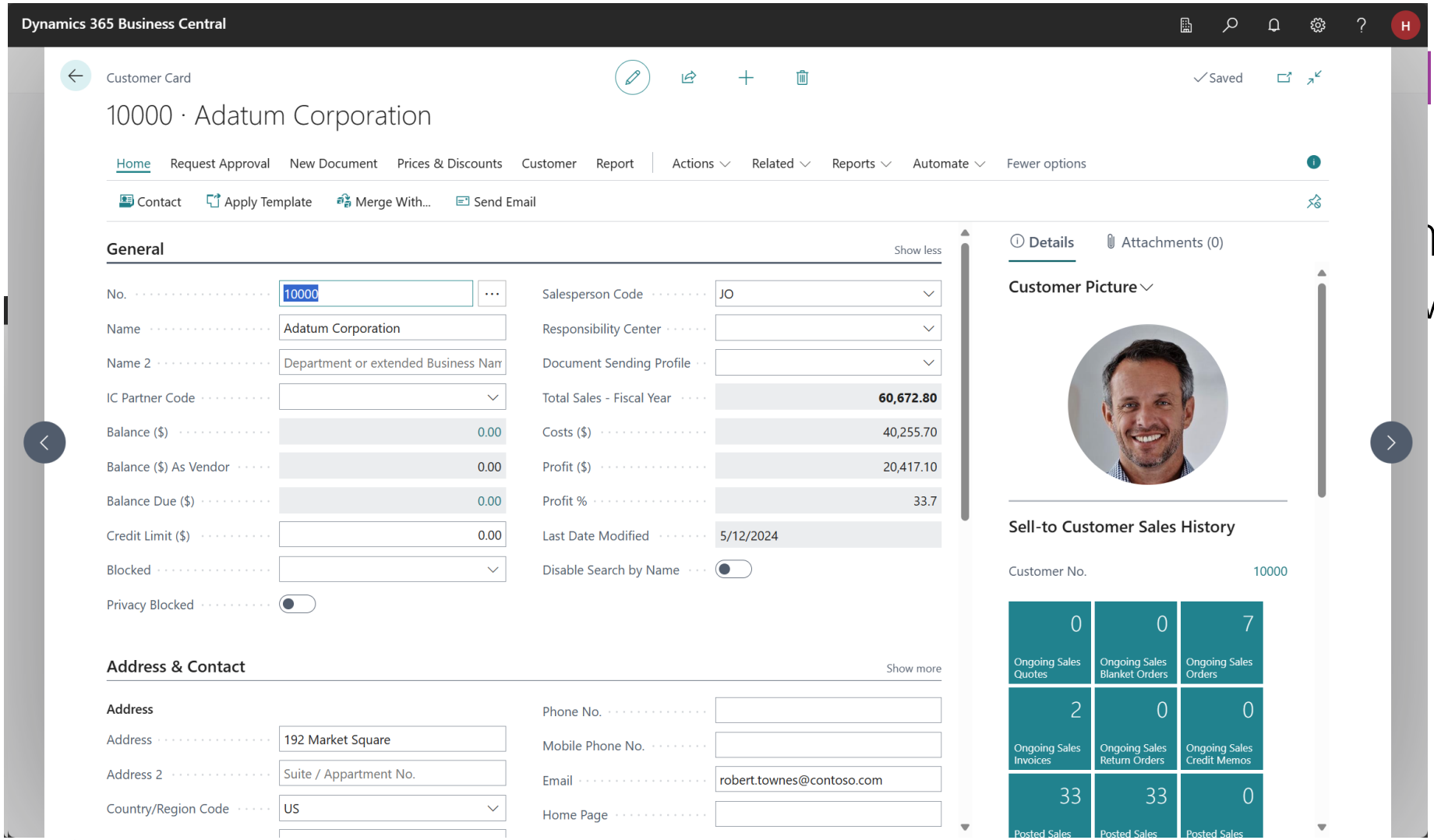
```
ErrorDialogWithTwoActions.Title := 'The line dimension value isn''t valid';
    ErrorDialogWithTwoActions.Message := StrSubstNo('The dimension value must be blank for the dimension %1 for Vendor %2', DimensionCode, VendorCode);
    ErrorDialogWithTwoActions.DetailedMessage('Add some text to help the person troubleshooting this error.');

    ErrorDialogWithTwoActions.AddAction(
        'Set value to blank',
        Codeunit::FirstFixitCodeunit,
        "FirstFixitCodeunitMethodName"
    );
    ErrorDialogWithTwoActions.AddAction(
        'OK',
        Codeunit::SecondFixitCodeunit,
        "SecondFixitCodeunitMethodName"
    );

    Error(ErrorDialogWithTwoActions);
```

Actionable errors References

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-error-handling-guidelines

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-actionable-errors

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-actionable-errors#error-messages-with-multiple-actions---how-to-use-them



es

nany other
w create a
that is shown in

Give users a hint for field values

Property: **`InstructionalText`**

Sets the string used for instructions in the UI.

The default is an empty string, which means there are no instructions. According to the user assistance model for Business Central, apps are expected to apply instructional text, also called placeholder text, to setup guides and similar pages.

Note

The InstructionalText property can be applied to Text, BigText, Guid, and Code data type fields.

https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/properties/devenv-instructionaltext-property

Follow Standard Nomenclature

When naming fields, make sure to use nomenclature the user will know.

Avoid:

- ID – i.e. Customer ID
- Number
- Synonyms – i.e. Use Customer, not Client. Item, not Material.

Follow Standard Nomenclature

The same rule pertains to Actions. Adding Groups for your app – especially if they contain actions that ‘fit’ in the existing groups, should be avoided.

Adding a arbitrary App Identifier to your actions, will ‘Scream’ “I am part of a separate app” – just like if You put a yellow driver side door onto your red car.

A user should NOT be able to tell ‘Standard’ and ‘App’ apart.

Follow Standard Nomenclature

Master Tables

- No. – Code[20]
- Has Number Series

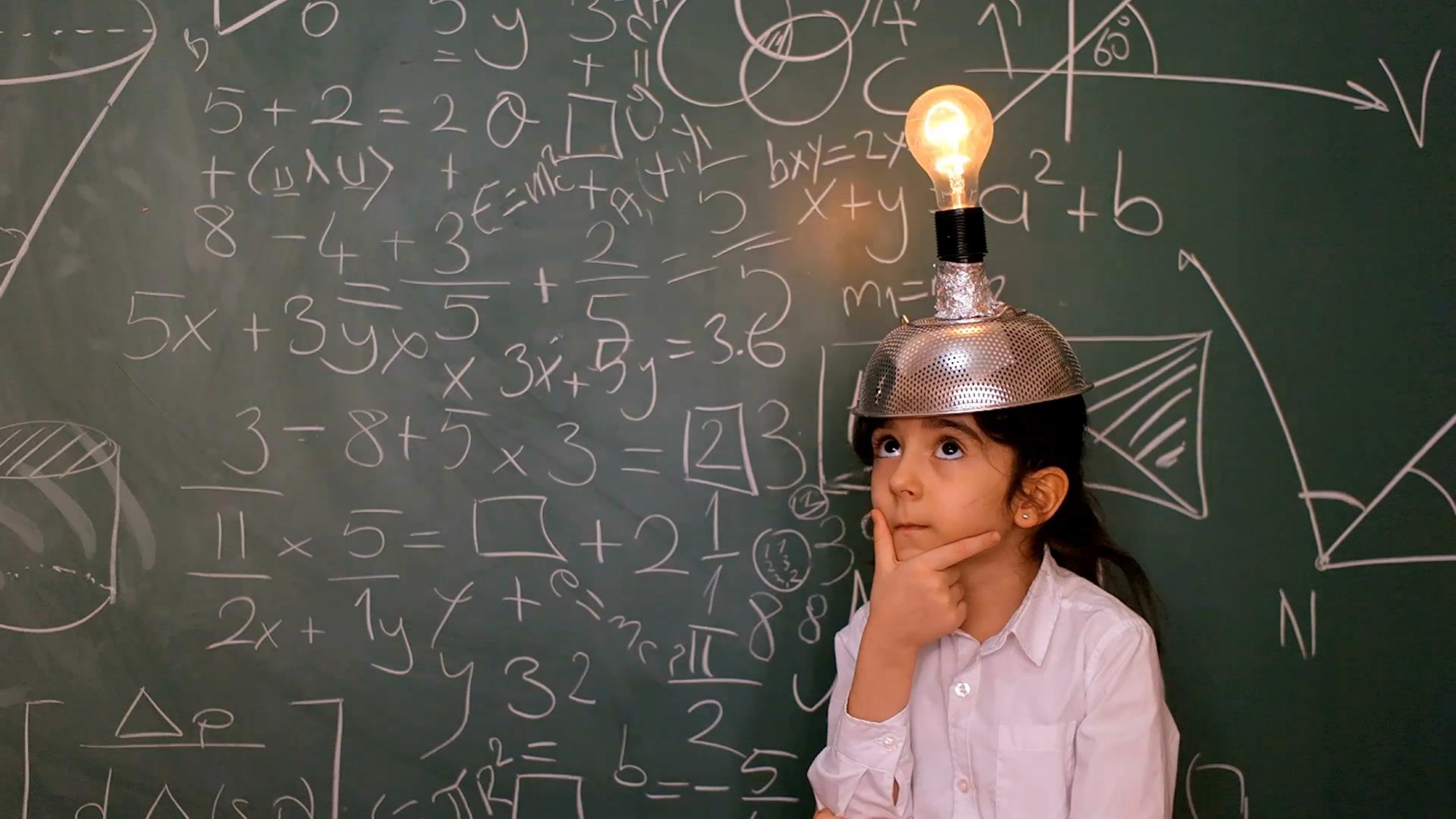
Support tables:

- Code - Code[10]
- Description Text[50]

Follow Standard Nomenclature

Visit

https://alguidelines.dev


5 + 2 = 20
8 − 4 + 3
5x + 3y
3 − 8 + 5 × 3 = 23
11 × 5 = □ + 2 1/1
2x + 1y
E=mc²
x + y
a² + b
60°

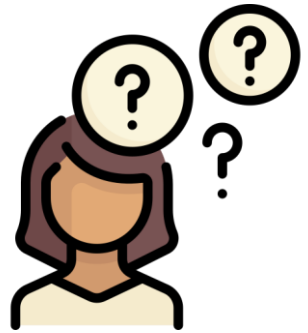
Acknowledgments

Most code examples shown here, are retrieved, thanks to Stefan Maron and his MSDyn365BC.Code.History repository on GitHub

https://github.com/StefanMaron/MSDyn365BC.Code.History

Questions?

- Twitter: @TheDoubleH
- Email: henrik@helgesenconsulting.com
- Blog: https://TheDoubleH.dev
- Github: https://github.com/TheDoubleH



Thank You