C/AL Coding Guidelines used at Microsoft Development Center Copenhagen

We are publishing our current C/AL coding guidelines.

The plan is to simply expose what we use now. And more important, to say that guidelines could be used. Debate on individual guidelines can become heated for any programming language, but the benefit of using some or other guidelines stays.

For us, those guidelines are enforced at check-in time - we are using a tool which verifies and only allows compliant check-ins. While this tool is internal and not ready to publish, we had anyways decided to open up and present the rules we use to the community, as inspiration.

Question: Since we're having the guidelines, how come there is still C/AL code in NAV which doesn't respect them?

Answer: all new C/AL code is bound to follow the guidelines (else it cannot be checked-in). However, the code that existed before the rules - it does not. We had done cleanup in a certain degree. Now we're gradually improving the old code base as we visit various objects in order to add new functionality, however chances are that code we didn't touch in a long time had remained in its old form.

You can comment on the rules on the NAV Dynamics Community Wiki at:

https://community.dynamics.com/nav/w/designpatterns/156.cal-coding-guidelines.aspx

Best regards,

The NAV Design Patterns team

Design

Parameter placeholders

The number of parameters passed to a string must match the placeholders.

Bad code

```
1. CannotDeleteLineErr@1005 : TextConst 'ENU=You cannot delete this line
   because one or more rating values exists.';
2. ...
3. ERROR(CannotDeleteLineErr,TABLECAPTION);
```

Good code

```
1. CannotDeleteLineErr@1005 : TextConst 'ENU=You cannot delete this line
   because one or more rating values exists.';
2. ...
3. ERROR(CannotDeleteLineErr);
```

Bad code

```
1. CannotUseThisFieldErr@1020 : TextConst 'ENU=You cannot use this field
   for %2 fields.';
2. ...
3. ERROR(CannotUseThisFieldErr,0,Field.Class);
```

Good code

```
1. CannotUseThisFieldErr@1020 : TextConst 'ENU=You cannot use this field
   for %1 fields.';
2. ...
```

```
3.  ERROR(CannotUseThisFieldErr,Field.Class);
```

Unreachable code

Do not write code that will never be hit.

It affects code readability and can lead to wrong assumptions.

Bad code

```
1.  IF Type <> Type::FIELD THEN BEGIN
2.    ...
3.    ERROR(...);
4.    RecRef.CLOSE;
5.  END;
```

Good code

```
1.  IF Type <> Type::FIELD THEN BEGIN
2.    ...
3.    RecRef.CLOSE;
4.    ERROR(...);
5.  END;
```

Unused variables

Do not declare variables that are unused.

Unused variables affect readability.

Bad code

```
1.  PROCEDURE CheckPostingDate@23(CaptionEntryNo@1005 : Text[50]);
2.  BEGIN
```

```
3.    IF GenJnlCheckLine.DateNotAllowed(PostingDate) THEN
4.      ERROR(DateNotAllowedErr,Caption,EntryNo)
5.    IF PostingDate > MaxPostingDate THEN
6.      MaxPostingDate := PostingDate;
7.  END
```

Good code

```
1.  PROCEDURE CheckPostingDate@23();
2.  BEGIN
3.    IF GenJnlCheckLine.DateNotAllowed(PostingDate) THEN
4.      ERROR(DateNotAllowedErr,Caption,EntryNo);
5.    IF PostingDate > MaxPostingDate THEN
6.      MaxPostingDate := PostingDate;
7.  END;
```

Bad code

```
1.  PROCEDURE IsReturned@14(EntryNo@1002 : Integer) : Decimal;
2.  VAR
3.    ItemEntry@1000 : Record 32;
4.    Quantity@1003 : Integer;
5.  BEGIN
6.    EXIT(-OutboundApplied(EntryNo,TRUE) - InboundApplied(EntryNo,TRUE));
7.  END;
```

Good code

```
1.  PROCEDURE IsReturned@14(EntryNo@1002 : Integer) : Decimal;
2.  BEGIN
3.    EXIT(-OutboundApplied(EntryNo,TRUE) - InboundApplied(EntryNo,TRUE));
4.  END;
```

Variable capacity mismatch

Do not assign a value to a variable whose capacity is smaller.

It will throw an error at runtime.

Bad code

```
1. FileName@1010 : Text[250];
2. ...
3. UploadedFileName@1016 : Text[1024];
4. ...
5. FileName := UploadedFileName;
```

Good code

```
1. FileName@1010 : Text[1024];
2. ...
3. UploadedFileName@1016 : Text[1024];
4. ...
5. FileName := UploadedFileName;
```

Bad code

```
1. VAR
2.   ExceededNumberTxt@001 : 'ENU=Warning: Exceeded number of unsent
   documents/requests'
3.   Subject@1002 : Text[50];
4.   ...
5. BEGIN
6.   ...
7.   Subject := ExceededNumberTxt;
```

Good code

```
1. VAR
2.   ExceededNumberTxt@001 : 'ENU=Warning: Exceeded number of unsent
   documents/requests'
3.   Subject@1002 : Text[100];
4.   ...
```

```
5.  BEGIN
6.    ...
7.    Subject := ExceededNumberTxt';
```

WITH scope name collision

Do not use the WITH scope when it has a variable whose name is the same as a local variable. This can lead to wrong code assumptions.

Given that "Contract Type" is a field on table ServiceContractHeader, then in the following example there is a parameter name clash with the field name. Which one will be used?

Bad code

```
1.  PROCEDURE InsertData@1("Contract Type"@1000 : Option...);
2.   ...
3.  BEGIN
4.    ...
5.    WITH ServiceContractHeader DO BEGIN
6.      ...
7.      DimMgt.InsertServContractDim(...,"Contract Type","Contract
    No.",0,...);
8.    END;
```

Good code

```
1.  PROCEDURE InsertData@1(ContractType@1000 : Option...);
2.   ...
3.  BEGIN
4.    ...
5.    WITH ServiceContractHeader DO BEGIN
6.      ...
7.      DimMgt.InsertServContractDim(...,ContractType,"Contract No.",0,...);
8.    END;
```

Unused initialized variables

The value assigned to a variable must be used. Else the variable is not necessary.

Bad code

```
1.  PROCEDURE AddEntities@1(FilterStr@1000 : Text[250]);
2.  VAR
3.    Vendor@1001 : Record 23;
4.    Count@1002 : Integer;
5.  BEGIN
6.    Count := 0;
7.
8.    Vendor.SETFILTER("No.",FilterStr);
9.    IF Vendor.FINDSET THEN
10.     REPEAT
11.       "User ID" := USERID;
12.       "Vendor No." := Vendor."No.";
13.       IF INSERT THEN
14.         Count += 1;
15.     UNTIL Vendor.NEXT = 0;
16. END;
```

Good code

```
1.  PROCEDURE AddEntities@1(FilterStr@1000 : Text[250]);
2.  VAR
3.    Vendor@1001 : Record 23;
4.  BEGIN
5.    Vendor.SETFILTER("No.",FilterStr);
6.    IF Vendor.FINDSET THEN
7.      REPEAT
8.        "User ID" := USERID;
9.        "Vendor No." := Vendor."No.";
10.       IF INSERT THEN;
11.     UNTIL Vendor.NEXT = 0;
12. END;
```

Static object invocation

Call objects statically whenever possible. It reduces extra noise and removes extra variables. Downside: changing the name of the object which is called statically will need a code update.

Bad code

```
1.  LOCAL PROCEDURE Code@1();
2.  VAR
3.    CAJnlPostBatch@1001 : Codeunit 1103;
4.  BEGIN
5.    CAJnlPostBatch.Run(CostJnlLine);
6.  END;
```

Good code

```
1.  LOCAL PROCEDURE Code@1();
2.  BEGIN
3.    CODEUNIT.RUN(CODEUNIT::"CA Jnl.-Post Batch",CostJnlLine);
4.  END;
```

By reference parameters

Do not declare parameters by reference if their values are not intended to be changed.

Unintentional value changes might propagate. Also, it might lead people to believe that value changes are intended.

Bad code

```
1.  LOCAL PROCEDURE ShowMessage@15(VAR Text@1000 : Text[250]);
2.  BEGIN
3.    Text := GetMessageText;
4.    IF (Text <> '') AND GenJnlLineInserted THEN
5.      MESSAGE(Text);
6.  END;
```

Good code

```
1.  LOCAL PROCEDURE ShowMessage@15(Text@1000 : Text[250]);
2.  BEGIN
3.    Text := GetMessageText;
4.    IF (Text <> '') AND GenJnlLineInserted THEN
5.      MESSAGE(Text);
6.  END;
```

Initialized variables

Variables should always be set to a specific value.

Bad code

```
1.  PROCEDURE SetPurchLine@22(VAR CurrentPurchLine@1000 : Record 39);
2.  VAR
3.    Pegging@1001 : Boolean;
4.  BEGIN
5.    IF Pegging THEN
6.      CurrQuantity := CurrentPurchLine."Quantity (Base)"
7.    ELSE
8.      CurrQuantity := CurrentPurchLine."Outstanding Qty. (Base)";
9.  END;
```

Good code

```
1.  PROCEDURE SetPurchLine@22(VAR CurrentPurchLine@1000 : Record 39);
2.  VAR
3.    Pegging@1001 : Boolean;
4.  BEGIN
5.    Pegging := IsPegging(CurrentPurchLine);
6.    IF Pegging THEN
7.      CurrQuantity := CurrentPurchLine."Quantity (Base)"
8.    ELSE
```

```
9.      CurrQuantity := CurrentPurchLine."Outstanding Qty. (Base)";
10. END;
```

Bad code

```
1.  // In the example below, the function will always return FALSE.
2.  PROCEDURE GetItemsToPlan@22() : Boolean;
3.  BEGIN
4.    SETRANGE("Document Type","Document Type"::Order);
5.    ...
6.    FINDSET
7.  END;
```

Good code

```
1.  PROCEDURE GetItemsToPlan@22() : Boolean;
2.  BEGIN
3.    SETRANGE("Document Type","Document Type"::Order);
4.    ...
5.    EXIT(FINDSET)
6.  END;
```

Maintainability index

Maintainability Index: Do not write functions that have a very low maintainability index. This makes the code hard to maintain.

Bad code

```
1.  Any procedure / trigger that has a maintainability index < 20
```

Good code

```
1.  Any procedure / trigger that has a maintainability index >= 20.
2.  The Maintainability Index is computed as a function:
```

```
3.    - Lines Of Code
4.    - the Halstead Volume
5.    - Cyclomatic Complexity.
```

More info

- Halstead Volume
- Cyclomatic Complexity

Bad code

```
1. Any procedure / trigger that is > 100 lines of code
```

Good code

```
1. Any procedure / trigger that is <= 100 lines of code.
2. A full C/AL Statement counts as 1 line of code
```

Class Coupling

Do not write functions that have high class coupling. This makes the code hard to maintain.

Bad code

```
1. Any procedure / trigger that has class coupling of > 30
```

Good code

```
1. Any procedure / trigger that has class coupling of <= 30.
2. Class coupling is computed by summing the unique instances of the
   following in a code block:
3.    - every unique usage of a complex C/AL data type (table, codeunit,
   etc) as 1.
4.    - every unique usage of a DotNet type as 1.
```

Cyclomatic complexity

Do not write functions that have high cyclomatic complexity. This makes the code hard to maintain.

Bad code

```
1. Any procedure / trigger that has a cyclomatic complexity > 25, using the
   CC3 version mentioned in this article.
```

Good code

```
1. Any procedure / trigger that has a cyclomatic complexity <= 25, using
   the CC3 version.
2. The CC3 version is computed by summing the following in a code block:
3.    - each IF statement as 1.
4.    - each entire CASE as 1.
```

FINDSET, FINDFIRST, FINDLAST

FINDSET, FIND('+') or FIND('-') should only be used when NEXT is used and vice versa.

Bad code

```
1. IF Cust.FIND('-') THEN
2.    ERROR(CustIsBlockErr)
```

Good code

```
1. IF Cust.FINDFIRST THEN
2.    ERROR(CustIsBlockErr)
```

Bad code

```
1. IF Cust.FINDFIRST THEN
2.    REPEAT
3.      ...
```

```
4.    UNTIL Cust.NEXT = 0;
```

Good code

```
1.  IF Cust.FINDSET THEN
2.    REPEAT
3.      ...
4.    UNTIL Cust.NEXT = 0;
```

Encapsulate local functionality

Any function used local must be defined as local.

Internationalization

Using CALCDATE

CALCDATE should only be used with DateFormula variables. Alternatively the string should be enclosed using the <> symbols.

Bad code

```
1.  IF ReservEntry."Expected Receipt Date" >
2.     CALCDATE('-' + FORMAT("Dampener (Time)") + FirstDate
3.  THEN
```

Good code

```
1.  IF ReservEntry."Expected Receipt Date" >
       CALCDATE('<-' + FORMAT("Dampener (Time)") + FirstDate  + '>')
    THEN
```

Localizability

The localizability guidelines are recommended for solutions that will be exposed to more than one language.

Use Text Constants

Pass user messages using Text Constants. It makes translation easy.

Bad code

```
1.  ImportAttachmentQst@1021 : TextConst 'ENU="Import attachment "';
2.  ...
3.  IF CONFIRM(ImportAttachmentQst + Caption +'?',TRUE) THEN BEGIN
```

Good code

```
1.  ImportAttachmentQst@1021 : TextConst 'ENU="Import attachment %1?"';
2.  ...
3.  IF CONFIRM(STRSUBSTNO(ImportAttachmentQst, Caption),TRUE) THEN BEGIN
```

Bad code

```
1.  ...
2.      IF NOT
3.        CONFIRM(
4.          STRSUBSTNO(
5.            'Difference on Periodic entries: %1 on %2\'+
6.            'Do you want to continue?',Balance,Date),
7.          TRUE)
8.      THEN
9.        ERROR('Program terminated by the user');
```

Good code

```
1.  DiffOnPeriodEntiesQst@100 : TextConst 'ENU="Difference on Periodic
    entries: %1 on %2\ Do you want to continue?"';
```

```
2.  ProgramTerminatedErr@200 : TextConst 'ENU="Program terminated by the
    user"';
3.  ...
4.      IF NOT CONFIRM(STRSUBSTNO(DiffOnPeriodEntiesQst,Balance,Date),TRUE)
    THEN
5.        ERROR(ProgramTerminatedErr);
```

Global Text Constants

Declare Text Constant as global variables.

Bad code

```
1.  PROCEDURE GetRequirementText@6(...) : Text[50];
2.  VAR
3.    RequirementOptionsTxt@1002 : TextConst 'ENU=Shipment,Receive,Pick,Put-
    Away';
4.  BEGIN
```

Good code

```
1.  VAR
2.    RequirementOptionsTxt@1002 : TextConst 'ENU=Shipment,Receive,Pick,Put-
    Away';
3.  ...
4.  PROCEDURE GetRequirementText@6(...) : Text[50];
5.  BEGIN
```

Using OptionCaptionML

The OptionCaptionML should be filled in for sourceexpression using option data types.

Bad code

```
1.  { 30  ;TextBox      ;17850;0    ;150  ;423  ;Name=Selection;
2.                                          SourceExpr=Selection;
```

```
3.                                                  DataSetFieldName=Selection
    }
4.   ...
5.   Selection@1008 : 'Open,Closed,Open and Closed';
6.   ...
```

Good code

```
1.  { 30   ;TextBox      ;17850;0     ;150   ;423   ;Name=Selection;
2.                                                  OptionCaptionML=ENU=Open,
    Closed,Open and Closed;
3.                                                  SourceExpr=Selection;
4.                                                  DataSetFieldName=Selection
    }
5.   ...
6.   Selection@1008 : 'Open,Closed,Open and Closed';
```

CaptionML for system tables

CaptionML should always be specified on a page field for a system table. By default, system tables do not have captions, so if you need to use them in the UI then captions need to be added.

Bad code

```
1.   ...
2.  { 2    ;2    ;Field     ;
3.               SourceExpr=Name }
4.   ...
5.  OBJECT Table 2000000000 User
6.   ...
7.      { 2    ;    ;Name                 ;Text50          }
```

Good code

```
1.  OBJECT Page 123 User
2.   ...
3.  { 2    ;2    ;Field     ;
4.               CaptionML=ENU=Name;
```

```
5.              SourceExpr=Name }
6.   ...
7.  OBJECT Table 2000000000 User
8.   ...
9.      { 2   ;   ;Name                ;Text50          }
```

FIELDCAPTION and TABLECAPTION

For user messages, errors etc., use FIELDCAPTION not FIELDNAME and TABLECAPTION not TABLENAME.

Reason:

1. The correct translation will be automatically used.
2. If the caption/name changes, then there will be a single point of change needed.

Bad code

```
1.  IF NOT CONFIRM(STRSUBSTNO(UpdateLocationQst,FIELDNAME("Location
    Code"),...))
```

Good code

```
1.  IF NOT CONFIRM(STRSUBSTNO(UpdateLocationQst,FIELDCAPTION("Location
    Code"),...))
```

Readability

Generally, all readability rules are Microsoft style choices only. You can use them to keep consistency with the existing code.

Spacing: binary operators

There must be exactly one space character on each side of a binary operator such as = + - AND OR =. The parameter comma operator however, should have no spaces.

Bad code

```
1.  "Line Discount %" := "Line Discount Amount"/"Line Value"*100
```

Good code

```
1.  "Line Discount %" := "Line Discount Amount" / "Line Value" * 100;
```

Bad code

```
1.  StartDate := CALCDATE('<+'+FORMAT(Days + i)+'D>', StartDate);
```

Good code

```
1.  StartDate := CALCDATE('<+' + FORMAT(Days + i) + 'D>',StartDate);
```

Bad code

```
1.  StartDate := 0D;    // Initialize
```

Good code

```
1.  StartDate := 0D; // Initialize
```

Spacing: unary operators

There must be no space between a unary operator and its argument (except for the NOT keyword).

Bad code

```
1.  IF NOT(Type = Type::Item) THEN
```

Good code

```
1.  IF NOT (Type = Type::Item) THEN
```

Bad code

```
1.  DiscAmt := - "Discount Amount";
```

Good code

```
1.  DiscAmt := -"Discount Amount";
```

Spacing: brackets and ::

There must be no spaces characters before and after [] dimension brackets symbols or :: option symbols.

Bad code

```
1.  A[i] [j] := Amt;
```

```

```

Good code

```
1.  A[i][j] := Amt;
```

Bad code

```
1.  "Currency Exchange Rate"."Fix Exchange Rate Amount" :: Currency:
```

Good code

```
1.  "Currency Exchange Rate"."Fix Exchange Rate Amount"::Currency:
```

Bad code

```
1.  IF FIND (Which) THEN
```

Good code

```
1.  IF FIND(Which) THEN
```

Blank lines

Do not use blank lines at the beginning or end of any functions, after BEGIN, before END, or inside multiline expressions.

Bad code

```
1.  PROCEDURE MATRIX_OnDrillDown@1133(MATRIX_ColumnOrdinal : Integer);
```

```
2.  BEGIN
3.
4.    SetupDrillDownCol(MATRIX_ColumnOrdinal);
5.    DrillDown(FALSE,ValueType);
6.
7.  END;
```

Good code

```
1.  PROCEDURE MATRIX_OnDrillDown@1133(MATRIX_ColumnOrdinal : Integer);
2.  BEGIN
3.    SetupDrillDownCol(MATRIX_ColumnOrdinal);
4.    DrillDown(FALSE,ValueType);
5.  END;
```

Bad code

```
1.  IF NameIsValid AND
2.
3.     Name2IsValid
4.  THEN
```

Good code

```
1.  IF NameIsValid AND
2.     Name2IsValid
3.  THEN
```

BEGIN..END

Only use BEGIN..END to enclose compound statements.

Bad code

```
1.  IF FINDSET THEN BEGIN
```

```
2.    REPEAT
3.      ...
4.    UNTIL NEXT = 0;
5.  END;
```

Good code

```
1.  IF FINDSET THEN
2.    REPEAT
3.      ...
4.    UNTIL NEXT = 0;
```

Bad code

```
1.  IF IsAssemblyOutputLine THEN BEGIN
2.    TESTFIELD("Order Line No.",0);
3.  END;
```

Good code

```
1.  Good
2.  IF IsAssemblyOutputLine THEN
3.    TESTFIELD("Order Line No.",0);
```

Bad code

```
1.  IF FINDSET THEN
2.    REPEAT
3.      BEGIN
4.        ...
5.      END;
6.    UNTIL NEXT = 0;
```

Good code

```
1.  IF FINDSET THEN
2.    REPEAT
3.      ...
4.    UNTIL NEXT = 0;
```

Exception

```
1.  // Except for this case
2.
3.  IF X THEN BEGIN
4.    IF Y THEN
5.      DO SOMETHING;
6.  END ELSE (not X)
```

Unnecessary TRUE/FALSE

Do not use TRUE or FALSE keywords unnecessarily if the expression is already an logical expression.

Bad code

```
1.  IF IsPositive() = TRUE THEN
```

Good code

```
1.  IF IsPositive THEN
```

Bad code

```
1.  IF Complete <> TRUE THEN
```

Good code

```
1.  IF NOT Complete THEN
```

Unnecessary Function Paranthesis

Do not use parenthesis in a function call if the function does not have any parameters.

Bad code

```
1.  IF ReservMgt.IsPositive() THEN
```

Good code

```
1.  IF ReservMgt.IsPositive THEN
```

Bad code

```
1.  IF ChangeStatusForm.RUNMODAL() <> ACTION::Yes THEN
```

Good code

```
1.  IF ChangeStatusForm.RUNMODAL <> ACTION::Yes THEN
```

Unnecessary Compound Parenthesis

Use parenthesis only to enclose compound expressions inside compound expressions.

Bad code

```
1.  IF ("Costing Method" = "Costing Method"::Standard) THEN
```

```

```

Good code

```
1.  IF "Costing Method" = "Costing Method"::Standard THEN
```

1.1.1.1

Bad code

```
1.  ProfitPct = -(Profit) / CostAmt * 100;
```

Good code

```
1.  ProfitPct = -Profit / CostAmt * 100;
```

Comments: curly brackets

Never use curly bracket comments. During development, the “Block comment” functionality can be used instead. However, in production code, block comments are not recommended.

Bad code

```
1.  PeriodTxt: {Period}
```

Good code

```
1.  PeriodTxt: // Period
```

Bad code

```
1.  PROCEDURE MATRIX_OnAfterGetRecord@10(MATRIX_ColumnOrdinal : Integer);
```

```
2.  BEGIN
3.    {
4.    IF ShowColumnName THEN
5.      MatrixHeader := MatrixRecords[MATRIX_ColumnOrdinal].Name
6.    ELSE
7.      MatrixHeader := MatrixRecords[MATRIX_ColumnOrdinal].Code;
8.    }
9.    MatrixRecord := MatrixRecords[MATRIX_ColumnOrdinal];
10.   AnalysisValue := CalcAmt(ValueType,TRUE);
11.   MATRIX_CellData[MATRIX_ColumnOrdinal] := AnalysisValue;
12. END;
```

Good code

```
1.  PROCEDURE MATRIX_OnAfterGetRecord@10(MATRIX_ColumnOrdinal : Integer);
2.  BEGIN
3.    MatrixRecord := MatrixRecords[MATRIX_ColumnOrdinal];
4.    AnalysisValue := CalcAmt(ValueType,TRUE);
5.    MATRIX_CellData[MATRIX_ColumnOrdinal] := AnalysisValue;
6.  END;
```

Comments: spacing

Always start comments with // followed by one space character.

Bad code

```
1.  RowNo += 1000; //Move way below the budget
```

Good code

```
1.  RowNo += 1000; // Move way below the budget
```

Indentation

In general, use an indentation of two space characters. Logical expressions in the IF, WHILE, and UNTIL parts are indented at least 3, 6, and 6 spaces respectively.

Bad code

```
1.  IF GLSetup."Unrealized VAT" OR
2.    (GLSetup."Prepayment Unrealized VAT" AND NewCVLedgEntryBuf.Prepayment)
```

Good code

```
1.  IF GLSetup."Unrealized VAT" OR
2.     (GLSetup."Prepayment Unrealized VAT" AND
    NewCVLedgEntryBuf.Prepayment)
```

Bad code

```
1.  IF GenJnlLine."Account No." <> ICPartner.Code THEN
2.    ICPartner.GET("Account No.");
3.    IF GenJnlLine.Amount > 0 THEN BEGIN
4.      ...
```

Good code

```
1.  IF GenJnlLine."Account No." <> ICPartner.Code THEN
2.    ICPartner.GET("Account No.");
3.  IF GenJnlLine.Amount > 0 THEN BEGIN
4.    ...
```

Bad code

```
1.  Dialog.OPEN(WindowTxt +
2.              '@1@@@@@@@@@@@@@@@@@@@@@@@@@');
```

Good code

```
1.  Dialog.OPEN(
2.    WindowTxt +
3.    '@1@@@@@@@@@@@@@@@@@@@@@@@@@');
```

Bad code

```
1.  TempOldCustLedgEntry.DELETE;
2.    // Find the next old entry for application of the new entry
```

Good code

```
1.  TempOldCustLedgEntry.DELETE;
2.  // Find the next old entry for application of the new entry
```

Bad code

```
1.  IF NOT ("Applies-to Doc. Type" IN
2.     ["Applies-to Doc. Type"::Receipt,
3.      "Applies-to Doc. Type"::"Return Shipment"])
```

Good code

```
1.  IF NOT ("Applies-to Doc. Type" IN
2.           ["Applies-to Doc. Type"::Receipt,
3.            "Applies-to Doc. Type"::"Return Shipment"])
```

Bad code

```
1.  WHILE (RemAmt > 0) OR
2.    (RemAmtLCY > 0)
```

```
3.  DO
```

Good code

```
1.  WHILE (RemAmt > 0) OR
2.         (RemAmtLCY > 0)
3.  DO
```

Bad code

```
1.  UNTIL (RemAmt > 0) AND
2.    (RemAmtLCY > 0);
```

Good code

```
1.  UNTIL (RemAmt > 0) AND
2.         (RemAmtLCY > 0)
```

BEGIN as an afterword

When BEGIN follows THEN, ELSE, DO, it should be on the same line, preceded by one space character.

Bad code

```
1.  IF ICPartnerRefType = ICPartnerRefType::"Common Item No." THEN
2.    BEGIN
3.    ...
4.  END;
```

Good code

```
1.  IF ICPartnerRefType = ICPartnerRefType::"Common Item No." THEN BEGIN
2.    ...
3.  END;
```

CASE actions

A CASE action should start on a line after the possibility.

Bad code

```
1.  CASE Letter OF
2.    'A': Letter2 := '10';
3.    'B': Letter2 := '11';
```

Good code

```
1.  CASE Letter OF
2.    'A':
3.      Letter2 := '10';
4.    'B':
5.      Letter2 := '11';
```

Separate IF and ELSE

IF and ELSE statements should be on separate lines.

Bad code

```
1.  IF Atom[i+1] = '>' THEN HasLogicalOperator := TRUE ELSE BEGIN
2.    ...
3.  END;
```

Good code

```
1.  IF Atom[i+1] = '>' THEN
2.     HasLogicalOperator := TRUE
3.  ELSE BEGIN
4.    ...
5.  END;
```

Lonely REPEAT

The REPEAT statement should always be alone on a line.

Bad code

```
1.  IF ReservEntry.FINDSET THEN REPEAT
```

Good code

```
1.  IF ReservEntry.FINDSET THEN
2.    REPEAT
```

Line-start keywords

The END, IF, REPEAT, FOR, WHILE, ELSE and CASE statement should always start a line.

Bad code

```
1.  IF IsContactName THEN ValidateContactName
2.    ELSE IF IsSalespersonCode THEN ValidateSalespersonCode
3.      ELSE IF IsSalesCycleCode THEN ValidatSalesCycleCode;
```

Good code

```
1.  IF IsContactName THEN
2.    ValidateContactName
3.  ELSE
4.    IF IsSalespersonCode THEN
5.      ValidateSalespersonCode
6.    ELSE
7.      IF IsSalesCycleCode THEN
8.        ValidatSalesCycleCode;
```

Colon usage in CASE

The last possibility on a CASE statement must be immediately followed by a colon.

Bad code

```
1. CASE DimOption OF
2.   DimOption::"Global Dimension 1" :
3.     DimValue."Dimension Code" := GLSetup."Global Dimension 1 Code";
```

Good code

```
1. CASE DimOption OF
2.   DimOption::"Global Dimension 1":
3.     DimValue."Dimension Code" := GLSetup."Global Dimension 1 Code";
```

Variables declarations order

Variables declarations should be ordered by type. In general, object and complex variable types are listed first followed by simple variables. The order should be the same as the object list in the object designer for C/AL objects. Afterwards come the complex variables like RecordRef, .NET, FieldRef etc. At the end come all the simple data types in no particular order.

Bad code

```
1. StartingDateFilter@1002 : Text[30];
2. Vend@1003 : Record 23;
```

Good code

```
1. Vend@1003 : Record 23;
2. StartingDateFilter@1002 : Text[30];
```

Keyword pairs indentation

The IF..THEN pair, WHILE..DO pair, and FOR..DO pair must appear on the same line or the same level of indentation.

Bad code

```
1. IF (x = y) AND
2.    (a = b) THEN
```

Good code

```
1.  IF (x = y) AND
2.     (a = b)
3.  THEN
```

END ELSE pair

The END ELSE pair should always appear on the same line.

Bad code

```
1.  IF OppEntry.FIND('-') THEN
2.    IF SalesCycleStage.FIND('-') THEN BEGIN
3.      ...
4.    END
5.  ELSE
6.    ...
```

Good code

```
1.  IF OppEntry.FIND('-') THEN
2.    IF SalesCycleStage.FIND('-') THEN BEGIN
3.      ...
4.    END ELSE
5.      ...
```

One statement per line

A line of code should not have more than one statement.

Bad code

```
1.  IF OppEntry.FIND('-') THEN EXIT
```

Good code

```
IF OppEntry.FIND('-') THEN
  EXIT
```

Bad code

```
TotalCost += Cost; TotalAmt += Amt;
```

Good code

```
TotalCost += Cost;
TotalAmt += Amt;
```

Unnecessary separators

There should be no unnecessary separators.

Bad code

```
IF Customer.FINDFIRST THEN;;
```

Good code

```
IF Customer.FINDFIRST THEN;
```

Unnecessary ELSE

ELSE should not be used when the last action in the THEN part is an EXIT, BREAK, SKIP, QUIT, ERROR.

Bad code

```
1.  IF IsAdjmtBinCodeChanged THEN
2.    ERROR(AdjmtBinCodeChangeNotAllowedErr,...)
3.  ELSE
4.    ERROR(BinCodeChangeNotAllowedErr,...);
```

Good code

```
1.  IF IsAdjmtBinCodeChanged THEN
2.    ERROR(AdjmtBinCodeChangeNotAllowedErr,...)
3.  ERROR(BinCodeChangeNotAllowedErr,...);
```

Nested WITHs

Do not nest WITHs that reference different types of objects.

Bad code

```
1.  WITH PostedWhseShptLine DO BEGIN
2.    ...
3.    WITH ItemLedgEntry DO
4.      InsertBufferRec(...,"Serial No.","Lot No.",...);
5.      ...
6.  END;
```

Good code

```
1.  WITH PostedWhseShptLine DO BEGIN
2.    ...
3.    InsertBufferRec(...,ItemLedgEntry."Serial No.",ItemLedgEntry."Lot
    No.",...);
4.    ...
5.  END;
```

Quotation marks in naming

Blanks,periods and other characters that would require quotation marks around a variable must be omitted. Else when the code is read, they are easily mistaken for fields.

Bad code

```
1.  LOCAL PROCEDURE HandleCustDebitCredit@17(...;"Amount (LCY)"@1001 :
    Decimal;...);
2.  BEGIN
3.    IF ((... ("Amount (LCY)" > 0)) ...) OR
4.       ((... ("Amount (LCY)" < 0)) ...)
5.    THEN BEGIN
6.    ...
```

Good code

```
1.  LOCAL PROCEDURE HandleCustDebitCredit@17(...;AmountLCY@1001 :
    Decimal;...);
2.  BEGIN
3.    IF ((... (AmountLCY > 0)) ...) OR
4.       ((... (AmountLCY < 0)) ...)
5.    THEN BEGIN
6.    ...
```

Variable already scoped

Do not use scope ''.'' qualifier unnecessarily when a variable is already implicitly or explicitly scoped. It keeps the code simpler.

Bad code

```
1.  ReturnRcptHeader.SETRANGE(ReturnRcptHeader."Return Order No.","Document
    No.");
```

Good code

```
1.  ReturnRcptHeader.SETRANGE("Return Order No.","Document No.");
```

Bad code

```
1. WITH ChangeLogSetupTable DO BEGIN
2.   ...
3.   IF ChangeLogSetupTable.DELETE THEN
4.     ...
5. END;
```

Good code

```
1. WITH ChangeLogSetupTable DO BEGIN
2.   ...
3.   IF DELETE THEN
4.     ...
5. END;
```

Binary operator line start

Do not start a line with a binary operator.

Bad code

```
1. "Quantity to Ship" :=
2.   Quantity
3.   - "Quantity Shipped"
```

Good code

```
1. "Quantity to Ship" :=
2.   Quantity -
3.   "Quantity Shipped"
```

Unary operator line end

Do not end a line with unary operator.

Bad code

```
1.  "Quantity Handled (Base)" := -
2.    "Quantity Handled (Base)");
```

Good code

```
1.  "Quantity Handled (Base)" :=
2.    - "Quantity Handled (Base)");
```

Named invocations

When calling an object statically use the name, not the number

Bad code

```
1.  PAGE.RUNMODAL(525,SalesShptLine)
```

Good code

```
1.  PAGE.RUNMODAL(PAGE::"Posted Sales Shipment Lines",SalesShptLine)
```

Variable naming

Variables that refer to a C/AL object must contain the objects name, abbreviated where necessary.

A variable must begin with a capital letter.

Blanks, periods, and other characters (such as parentheses) that would make quotation marks around a variable necessary must be omitted.

If a variable is a compound of two or more words or abbreviations, each word or abbreviation should begin with a capital letter.

Bad code

```
1. ...
2.     WIPBuffer@1002 : Record 1018
3. ...
4. OBJECT Table Job WIP Buffer
```

Good code

```
1. ...
2.     JobWIPBuffer@1002 : Record 1018
3. ...
4. OBJECT Table Job WIP Buffer
```

Bad code

```
1. ...
2.     Postline@1004 : Codeunit 12;
3. ...
4. OBJECT Codeunit Gen. Jnl.-Post Line
```

Good code

```
1. ...
2.     GenJnlPostLine@1004 : Codeunit 12;
3. ...
4. OBJECT Codeunit Gen. Jnl.-Post Line
```

Temporary variable naming

The name of a temporary variable must be prefixed with the word Temp and not otherwise.

Bad code

```
1. JobWIPBuffer@1002 : TEMPORARY Record 1018;
```

```

```

Good code

```
1.  TempJobWIPBuffer@1002 : TEMPORARY Record 1018;
```

Bad code

```
1.  TempJobWIPBuffer@1002 : Record 1018;
```

Good code

```
1.  CopyOfJobWIPBuffer@1002 : Record 1018;
```

TextConst suffixes

TextConst variable names should have a suffix (an approved three-letter suffix: Msg, Tok, Err, Qst, Lbl, Txt) describing usage.

Bad code

```
1.  CannotDeleteLine@1005 : TextConst 'ENU=You cannot delete this line
    because one or more rating values exists.';
2.  ...
3.  ERROR(CannotDeleteLine,TABLECAPTION);
```

Good code

```
1.  CannotDeleteLineErr@1005 : TextConst 'ENU=You cannot delete this line
    because one or more rating values exists.';
2.  ...
3.  ERROR(CannotDeleteLineErr,TABLECAPTION);
```

Bad code

```
1.  Text000@1011 : TextConst 'ENU="has been changed (initial a %1: %2= %3,
    %4= %5)"';
2.  ...
3.  SalesLine.FIELDERROR(Type,STRSUBSTNO(Text000,...);
4.  ...
```

Good code

```
1.  TypeHasBeenChangedErr@1011 : TextConst 'ENU="has been changed (initial a
    %1: %2= %3, %4= %5)"';
2.  ...
3.  SalesLine.FIELDERROR(Type,STRSUBSTNO(TypeHasBeenChangedErr,...);
4.  ...
```

Bad code

```
1.  Text004@1004 : TextConst 'ENU=Indenting the Job Tasks #1##########.';
2.  ...
3.  Window@1007 : Dialog;
4.  ...
5.    Window.OPEN(Text004);
```

Good code

```
1.  IndentingMsg@1004 : TextConst 'ENU=Indenting the Job Tasks
    #1##########.';
2.  ...
3.  Window@1007 : Dialog;
4.  ...
5.    Window.OPEN(IndentingMsg);
```

Bad code

```
1. Text002@1005 : TextConst 'ENU=You cannot delete a %1 that is used in one
   or more setup windows.\ Do you want to open the G/L Account No. Where-
   Used List Window?';
2. ...
3. IF CONFIRM(Text002,TRUE,GLAcc.TABLECAPTION) THEN
```

Good code

```
1. OpenWhereUsedWindowQst@1005 : TextConst 'ENU=You cannot delete a %1 that
   is used in one or more setup windows.\ Do you want to open the G/L
   Account No. Where-Used List Window?';
2. ...
3. IF CONFIRM(OpenWhereUsedWindowQst,TRUE,GLAcc.TABLECAPTION) THEN
```

Bad code

```
1. Selection := STRMENU(Text003,2);
2. ...
3. Text003@1002 : TextConst 'ENU=&Copy dimensions from BOM,&Retrieve
   dimensions from components';
```

Good code

```
1. Selection := STRMENU(CopyFromQst,2);
2. ...
3. CopyFromQst@1002 : TextConst 'ENU=&Copy dimensions from BOM,&Retrieve
   dimensions from components';
```

Bad code

```
1. DATASET
2. {
3. ...
4.    { 1   ;1   ;Column  ;Chart_of_AccountsCaption;
```

```
5.                              SourceExpr=Chart_of_AccountsCaption }
6.  ...
7.  Chart_of_AccountsCaption@9647 : TextConst 'ENU=Chart of Accounts';
```

Good code

```
1.  DATASET
2.  {
3.  ...
4.    { 1   ;1   ;Column  ;Chart_of_AccountsCaption;
5.                              SourceExpr=ChartOfAccountsLbl }
6.  ...
7.  ChartOfAccountsLbl@9647 : TextConst 'ENU=Chart of Accounts';
```

User eXperience

MESSAGE and ERROR

Always end MESSAGE or ERROR with a period.

Bad code

```
1. CustIsBlockedErr@1025 : TextConst 'ENU=You cannot %1 this type of
   document when Customer %2 is blocked with type %3';
2. ...
3. ERROR(CustIsBlockedErr,...);
```

Good code

```
1. CustIsBlockedErr@1025 : TextConst 'ENU=You cannot %1 this type of
   document when Customer %2 is blocked with type %3.';
2. ...
3. ERROR(CustIsBlockedErr,...);
```

FIELDERROR

Never use FIELDERROR with a period as it is automatically inserted.

Bad code

```
1. InvalidValue@1025 : TextConst 'ENU=is invalid.';
2. ...
3. Cust.FIELDERROR("No.",InvalidValue);
```

Good code

```
1. InvalidValue@1025 : TextConst 'ENU=is invalid';
2. ...
3. Cust.FIELDERROR("No.",InvalidValue);
```

CONFIRM

Always end CONFIRM with a question mark.

Bad code

```
1. ChangeAllOpenedEntriesQst@1000 : TextConst 'ENU=Do you want to change
   all open entries for every customer and vendor that are not blocked';
2. ...
3. IF CONFIRM(ChangeAllOpenedEntriesQst,TRUE) THEN
```

Good code

```
1. ChangeAllOpenedEntriesQst@1000 : TextConst 'ENU=Do you want to change
   all open entries for every customer and vendor that are not blocked?';
2. ...
3. IF CONFIRM(ChangeAllOpenedEntriesQst,TRUE) THEN
```

Actions images

All actions must have an image assigned to them.

Bad code

```
1. { 7       ;1   ;Action    ;
2.                CaptionML=ENU=Customer - &Balance;
3.                RunObject=Report 121 }
```

Good code

```
1. { 7       ;1   ;Action    ;
2.                CaptionML=ENU=Customer - &Balance;
3.                RunObject=Report 121 }
4.                Image=Report }
```

Suggested Abbreviations

Whenever possible, do not use abbreviations in variables, functions and objects names.

If there is no other choice, then use the suggestions below.

Abbreviation	Full Text	Abbreviation	Full Text	Abbreviation	Full Text
Abs	absence	Avg	average	Class	classification
Acc	account	BA	ba db.	coll	collection
Acc	accounting	Bal	balance	col	column
Accum	accumulated	BOM	bill of materials	Cmt	comment
Act	action	Blnkt	blanket	Co	company
Activ	activity	Budg	budget	Comp	component
Add	additional	Buf	buffer	Cmpltn	completion
Addr	address	Bus	business	Comps	components
Adj	adjust	BIM	business interaction management	Compn	composition
Adjd	adjusted	Buy	buying	Compr	compression
Adjmt	adjustment	Calc	calculate	Concrnt	concurrent
Agrmt	agreement	Calcd	calculated	Conf	confidential
Alloc	allocation	Calcu	calculation	Cnfrmn	confirmation
Allow	allowance	Cal	calendar	Confl	conflict
Alt	alternative	Cap	capacity	Consol	consolidate
Amt	amount	CRP	capacity requirements planning	Consolid	consolidation
Amts	amounts	CF	cash flow	Consump	consumption
Ans	answer	CF	cashflow	Cont	contact
Appl	applies	ctlg	catalog	Cntr	container
Appln	application	Cat	category	Contr	contract
Arriv	arrival	CPU	Central Processing Unit	Contrd	contracted
Asm	assembly	Ctr	center	Ctrl	control
ATO	assemble to order	Chg	change	Ctrls	controls
Assgnt	assignment	Chgs	changes	Conv	conversion
Assoc	associated	Char	character	Cor	correction
Attmt	attachment	Chars	characters	Corres	correspondence
Auth	authorities	Chrg	charge	Corresp	corresponding
Auto	automatic	Chrgs	charges	Cst	cost
Avail	availability	Chk	check	COGS	sold
Cr	credit	Exec	execute	Integr	integration
Cumul	cumulate	Expd	expected	Int	interest

Curr	currency
Crnt	current
Cust	customer
CV	customer/vendor
Dly	daily
Damp	dampener
DBMS	database management system
D	date
Def	definition
Demo	demonstration
Dept	department
DP	department/project
Depr	depreciation
Desc	description
Dtl	detail
Dtld	detailed
Dtls	details
Dev	deviation
Diff	difference
Dim	dimension
Dir	direct
Disc	discount
Discr	discrete
Distr	distribute
Distrd	distributed
Distbtr	distributor
Distrn	distribution
Doc	document
Dupl	duplicate
Entrd	entered
Engin	engineering
Exch	exchange
Excl	excluding
No	number
Nos	numbers
Obj	object
Oper	operating
Exped	expedited
Exp	expense
Expr	expression
Expir	expiration
Ext	extended
Expl	explode
Expt	export
Fnl	final
Fin	finance
Fisc	fiscal
Fnshd	finished
FA	fixed asset
Fwd	forward
Frt	freight
Gen	general
GL	general ledger
Gr	group
Hdr	header
Hist	history
Hol	holiday
HR	human resource
ID	identification
Imp	import
Inbnd	inbound
Incl	including
Incld	included
Incmg	incoming
ISV	independent software vendor
Indust	industry
Info	information
Init	initial
Intra	Intrastat
Interact	interaction
Qty	quantity
Questn	questionnaire
Qte	quote
RF	radio frequency
Intm	Interim
IP	internal protocol
Invt	inventory
Invtbl	inventoriable
Inv	invoice
Invd	invoiced
IT	item tracking
Jnl	journal
Lang	language
Ledg	ledger
Lvl	level
Ln	line
Lt	list
LCY	local currency
Loc	location
Mail	mailing
Maint	maintenance
Mgt	management
Man	manual
Mfg	manufacturing
Mfr	manufacturer
Mat	material
Mktg	marketing
Max	maximum
Meas	measure
Msg	message
Min	minimum
Misc	miscellaneous
Mod	modify
Mth	month
Neg	negative
NonInvtbl	non-inventoriable
Notif	notification
Reserv	reservation
Resol	resolution
Res	resource
Rsp	response

Opp	opportunity
Ord	order
Ords	orders
Orig	original
Org	organization
Outbnd	outbound
Outg	Outgoing
Out	output
Outstd	outstanding
Ovhd	overhead
Pmt	payment
Pct	percent
Persnl	personnel
Phys	physical
Pic	picture
Plng	planning
Pstd	posted
Post	posting
Pos	positive
Prec	precision
Prepmt	prepayment
Prod	product
Prod	production
ProdOrd	production order
Proj	project
Prop	property
Prspct	prospect
Purch	purchase
Purch	purchases
Purchr	purchaser
PurchOrd	purchase order
Qlty	quality
SF	frequency
Stmt	statement
Stat	statistical
Stats	statistics
Stk	stock
SKU	stockkeeping unit
Rng	range
Rcpt	receipt
Rcd	received
Rec	record
Recs	records
Recncl	reconcile
Recon	reconciliation
Recur	recurring
Ref	reference
Reg	register
Regn	registration
Regd	registered
Rel	relation
Rels	relations
Rlshp	relationship
Rlse	release
Rlsd	released
Rem	remaining
Rmdr	reminder
Repl	replacement
Rplnsh	replenish
Rplnsht	replenishment
Rpt	report
Rep	represent
Repd	represented
Rqst	request
Reqd	required
Reqt	requirement
Reqts	requirements
Req	requisition
Rsv	reserve
Rsvd	reserved
Synch	synchronize
Temp	temporary
Tot	total
Transac	transaction
Trans	transfer
Transln	translation
Resp	responsibility
Rtn	retain
Rtnd	retained
Ret	return
Rets	returns
Revaln	revaluation
Rev	reverse
Rvw	review
Rnd	round
Rndd	rounded
Rndg	rounding
Rte	route
Rtng	routing
Rout	routine
Sales	sales & receivables
Saf	safety
Sched	schedule
Sec	second
Seg	segment
Sel	select
Selctn	selection
Seq	sequence
Ser	serial
SN	serial number
Serv	service
Sh	sheet
Shpt	shipment
Src	source
Spcl	special
Spec	specification
Specs	specifications
Std	standard
Whse	warehouse
WS	web shop
Wksh	worksheet
GL	g/l
Pct	%
Three-Tier	3-tier

Stm	stream	Trkg	tracking	Osynch	Outlook Synch
SQL	structured query language	Tblsht	troubleshoot		
Subcontr	subcontract	Tblshtg	troubleshooting		
Subcontrd	subcontracted	UOM	unit of measure		
Subcontrg	subcontracting	UT	unit test		
Sub	substitute	Unreal	unrealized		
Subst	substitution	Unrsvd	unreserved		
Sug	suggest	Upd	update		
Sugd	suggested	Valn	valuation		
Sugn	suggestion	Val	value		
Sum	summary	VAT	value added tax		
Suspd	suspended	Var	variance		
Sympt	symptom	Vend	vendor		

Internally used Dot Net Types

Dot Net Types

'mscorlib'.System.Convert

'mscorlib'.System.Globalization.CultureInfo

'mscorlib'.System.Globalization.DateTimeStyles

'mscorlib'.System.Globalization.NumberStyles

'mscorlib'.System.Type

'mscorlib'.System.Array

'mscorlib'.System.EventArgs

'mscorlib'.System.Security.Cryptography.SHA512Managed

'mscorlib'.System.Security.Cryptography.HashAlgorithm

'mscorlib'.System.Text.Encoding

'mscorlib'.System.Text.UTF8Encoding

'mscorlib'.System.Environment

'mscorlib'.System.IO.Directory

'mscorlib'.System.IO.Path

'mscorlib'.System.IO.File

'mscorlib'.System.IO.FileAttributes

'mscorlib'.System.Collections.ArrayList

'mscorlib'.System.Collections.IEnumerator

'mscorlib'.System.Collections.Generic.IEnumerator`1

'mscorlib'.System.TimeSpan

'mscorlib'.System.DateTime

'mscorlib'.System.DateTimeKind

'mscorlib'.System.DateTimeOffset

'mscorlib'.System.Decimal

'mscorlib'.System.String

'System'.System.Diagnostics.Process

'System'.System.Diagnostics.ProcessStartInfo

'System'.System.IO.Compression.CompressionMode

'System'.System.IO.Compression.GZipStream

'System'.System.Uri

'System'.System.UriPartial

'System.Data'.System.Data.DataColumn

'System.Data'.System.Data.DataTable

'System.Data'.System.Data.DataRow

'System.Web'.System.Web.HttpUtility

'System.Windows.Forms'.System.Windows.Forms.DialogResult

'System.Windows.Forms'.System.Windows.Forms.FileDialog

'System.Windows.Forms'.System.Windows.Forms.OpenFileDialog

'System.Windows.Forms'.System.Windows.Forms.SaveFileDialog

'System.Windows.Forms'.System.Windows.Forms.FolderBrowserDialog

'System.Xml'.*

'DocumentFormat.OpenXml'.*

'mscorlib'.System.IO.DirectoryInfo

'mscorlib'.System.IO.FileInfo

'Microsoft.Dynamics.Nav.Client.CodeViewerTypes'.Microsoft.Dynamics.Nav.Client.CodeViewerTypes.BreakpointCollection

'Microsoft.Dynamics.Nav.Client.CodeViewerTypes'.Microsoft.Dynamics.Nav.Client.CodeViewerTypes.VariableCollection

'Microsoft.Dynamics.Nav.SMTP'.Microsoft.Dynamics.Nav.SMTP.SmtpMessage

'Microsoft.Dynamics.Nav.Management.DSObjectPickerWrapper'.*

'Microsoft.Dynamics.Nav.Timer'.*

'Microsoft.Dynamics.Nav.DO.ClientProxyWrapper'.*

'Microsoft.Dynamics.Nav.Client.BusinessChart'.*

'Microsoft.Dynamics.Nav.Client.BusinessChart.Model'.*

'Microsoft.Dynamics.Nav.Integration.Office'.*

'Microsoft.Dynamics.Nav.Integration.Office.Mock'.*

'Microsoft.Dynamics.Nav.EwsWrapper'.*

'Microsoft.Dynamics.Nav.EwsWrapper.ALTestHelper'.*

'Microsoft.Dynamics.NAV.OLSync.OLSyncSupplier'.*

'Microsoft.Dynamics.Nav.OLSync.Common'.*

'Microsoft.Dynamics.Nav.NavUserAccount'.*

'Microsoft.Dynamics.Nav.OpenXml'.*

'Microsoft.Dynamics.Nav.RapidStart'.*

'Microsoft.Dynamics.Framework.RapidStart.Common'.*

'Microsoft.Dynamics.Nav.Client.TimelineVisualization'.Microsoft.Dynamics.Nav.Client.TimelineVisualization.VisualizationScenarios

'Microsoft.Dynamics.Framework.UI.WinForms.DataVisualization.Timeline'.Microsoft.Dynamics.Framework.UI.WinForms.DataVisualization.TimelineVisualization.DataModel+TransactionChangesRow

'Microsoft.Dynamics.Framework.UI.WinForms.DataVisualization.Timeline'.Microsoft.Dynamics.Framework.UI.WinForms.DataVisualization.TimelineVisualization.DataModel+TransactionChangesDataTable

'Microsoft.Dynamics.Framework.UI.WinForms.DataVisualization.Timeline'.Microsoft.Dynamics.Framework.UI.WinForms.DataVisualization.TimelineVisualization.DataModel+TransactionRow

'Microsoft.Dynamics.Framework.UI.WinForms.DataVisualization.Timeline'.Microsoft.Dynamics.Framework.UI.WinForms.DataVisualizat

ion.TimelineVisualization.DataModel+TransactionDataTable

'Microsoft.Office.Interop.Word'.*

'Microsoft.Office.Interop.Excel'.*

'Microsoft.Dynamics.BAPIWrapper'.*

'Microsoft.Dynamics.Nav.Types'.Microsoft.Dynamics.Nav.Types.ConfigSettings

'Microsoft.Dynamics.Nav.DocumentService'.*

'Microsoft.Dynamics.Nav.DocumentService.Types'.*

'mscorlib'.System.IO.StreamWriter

'Microsoft.Dynamics.Nav.Client.TimelineVisualization'.Microsoft.Dynamics.Nav.Client.TimelineVisualization.InteractiveTimelineVisualizationAddIn

'System'.System.ComponentModel.CancelEventArgs

'System'.System.Text.RegularExpressions.Regex

'System'.System.Text.RegularExpressions.RegexOptions

'mscorlib'.System.IO.StreamReader

'System.Windows.Forms'.System.Windows.Forms.Control

'System.Windows.Forms'.System.Windows.Forms.ControlEventArgs

'System.Windows.Forms'.System.Windows.Forms.DragEventArgs

'System.Windows.Forms'.System.Windows.Forms.GiveFeedbackEventArgs

'System.Windows.Forms'.System.Windows.Forms.HelpEventArgs

'System.Windows.Forms'.System.Windows.Forms.InvalidateEventArgs

'System.Windows.Forms'.System.Windows.Forms.KeyEventArgs

'System.Windows.Forms'.System.Windows.Forms.KeyPressEventArgs

'System.Windows.Forms'.System.Windows.Forms.LayoutEventArgs

'System.Windows.Forms'.System.Windows.Forms.MouseEventArgs

'System.Windows.Forms'.System.Windows.Forms.PaintEventArgs

'System.Windows.Forms'.System.Windows.Forms.PreviewKeyDownEventArgs

'System.Windows.Forms'.System.Windows.Forms.QueryAccessibilityHelpEventArgs

'System.Windows.Forms'.System.Windows.Forms.UICuesEventArgs