Microsoft Dynamics® AX

Best Practices for Microsoft Dynamics AX 2009 Development

White Paper

Date: April 16, 2010

Table of Contents

Best Practices for Microsoft Dynamics AX Development

When using the Microsoft Dynamics AX development environment, you should adhere to a set of best practices. The X++ compiler checks the code for best practice issues. These issues can result in best practice errors, warnings, or informational messages.
This guide covers Microsoft Dynamics AX application development best practices. They are recommended for any partner or end user who is enhancing or customizing Microsoft Dynamics AX.
These best practices apply to the following:

- Programming in the standard application
- Certified solutions

Microsoft Dynamics AX includes tools for checking best practices. Best practices that can be checked in this way are marked with the following symbols:

- 
- 
- 

Top Best Practices to Consider

The following sections provide guidance to avoid best practice violations. The sections provide guidance in the following areas:

General Best Practice Guidance

The following list provides general best practices guidance.

- Favor using positive logic.
- Use constants instead of numeric literals.
- Use enumerations in place of constants.
- Use explicit access modifiers.
- Do not use method parameters as an l-value.

Formatting Guidance

The following table provides formatting best practice guidance.

Best Practice	Example
Include a separating semicolon between the declarations and code.	`int I;` `;`

Include a blank line after the separating semicolon.	`int I;`
	`;`
	
	`I = 2;`

Place the opening brace at the beginning of the next line.	`if (someExpression)`
	`{`
	`  doSomething();`
	`}`

Align the closing brace with the corresponding opening brace.	`if (someExpression)`
	`{`
	`  doSomething();`
	`}`

Place the opening and closing braces each on their own line.	`if (someExpression)`
	`{`
	`  doSomething();`
	`}`

Best Practice	Example

Do not omit braces. Braces are not optional because they increase code readability and maintainability. They should be included, even for single statement blocks.	`if (someExpression)`
	`{`
	`  doSomething();`
	`}`

Omit braces for switch statements. These braces can be omitted because the case and break statements clearly indicate the beginning and ending.	`case 0:`
	`  doSomething();`
	`  break;`

Use a single space in the following cases: • On either side of the assignment operator.	Correct: `cust.Name = "Jo";` Incorrect: `cust.Name="Jo";`

• After the comma between parameters.	Correct:
	`public void doSomething(int _x, int _y)`
	Incorrect:
	`public void doSomething(int  x,int  y)`

• Between arguments.	Correct: `myAddress(myStr, 0, 1)` Incorrect: `myAddress(myStr,0,1)`
• Before flow control statements.	Correct: `while (x == y)` Incorrect: `while(x == y)`
• Before and after binary operators.	Correct: `if (x == y)` Incorrect: `if (x==y)`
• After the semicolon between the parts of a `for` statement.	Correct: `for (i = 0; i < 10; i++)` Incorrect: `for (i = 0;i < 10;i++)`
Do not use any spaces in the following cases: • After the opening or before the closing parenthesis.	Correct: `myAddress(myStr, 0, 1)` Incorrect: `myAddress( myStr, 0, 1 )`
• Between a member name and opening parenthesis.	Correct: `myAddress()` Incorrect: `myAddress ()`
• Before or after the brackets.	Correct: `x = dataArray[index];` Incorrect: `x = dataArray[ index ];`
• Before or after unary operators.	Correct: `if (!y)` Incorrect: `if (! y)`

Use four spaces as the standard indent. The tab key in code editor inserts four spaces. Indent in the following cases:
• The contents of code blocks.
	`if (someExpression)`
	`{`
	`  doSomething();`
	`}`

• Case blocks even though they do not use braces.	`switch (someExpression)`
	`{`
	`  case 0:`
	`  doSomething();`
	`  break;`
	`  …`
	`}`

• A wrapped line one indent from the previous line.	`lastAccount = this.doSomething(`
	`  cust,`
	`  firstAccount,`
	`  startDate,`
	`  endDate);`

Wrap lines that get too long to fit on a single line.	
Wrap shorter lines to improve clarity.	
Place each wrapped `select` and `while`	`select firstonly cust`

Best Practice	Example

`select` statement keyword at the beginning of a new line. The content associated with each keyword should be indented by one indent under the corresponding keyword.	`    where someExpression1`
	`        && someExpression2`
	`        && someExpression3;`
	
	`select count(RecId)`
	`    from cust`
	`    where someExpression1`
	`        && someExpression2`
	`        && someExpression3;`
	
	`while select firstonly cust`
	`    order by Name, AccountNum`
	`    where someExpression1`
	`        && someExpression2`
	`        && someExpression3`
	`{`
	`    …`
	`}`

Do not use more or less than four spaces to force special alignment.	Right
	`lastAccount = this.doSomething(`
	`    cust,`
	`    firstAccount,`
	`    startDate,`
	`    endDate);`
	Wrong (indent is 14 spaces)
	`last = this.do(`
	`              cust,`
	`              firstAccount,`
	`              startDate,`
	`              endDate);`

Put each indented parameter or argument on a separate line.	
Use switch statements over consecutive if statements.	
Do not use parenthesis around the value of the cases of a switch statement.	
Do not put the closing parenthesis for a method call on a new line.	

Naming Guidance

The following table provides naming best practice guidance.

Best Practice	Example
Use Camel case naming for member variables, method names, and local variables.	`serverClass;`
Use Pascal case naming for Application Object Tree (AOT) elements.	`AddressCountyRegion;`
Prefix parameter names with an underscore (_).	`myJob(Args _args)`
Do not use Hungarian notation. Do not encode the type of a variable in its name.	Incorrect: `strName`
Avoid prefixing local variables.	Incorrect: `stringName` or `intCount`
Use meaningful and self-documenting names.	

Commenting Code Guidance

This section provides best practice guidance for writing code comments. Comments should be used to describe the intent, algorithmic overview, and logical flow. Provide comments so that someone other than the original developer could understand the behavior and purpose of the code. It is a best practice that most code will have comments reflecting the developer intent and approach for the code. Use comments liberally. Include comments that indicate who made the changes, when the changes were made, why the changes were added, and what the changes do. Comments are particularly beneficial when multiple parties are involved in modifying and maintaining code. The following table provides code commenting best practice guidance.

Best Practice	Example
Do not use comments that repeat the code.	

Do not use multi-line syntax `/* … */` for comments. The single-line syntax `// …` is preferred even when a comment spans multiple lines.	`public int getCount()`
	`{`
	`  ;`
	
	`  // This comment spans multiple`
	`  // lines because it has`
	`  // a lot to say. The use of`
	`  // multi-line syntax is`
	`  // not allowed.`
	`  …`
	`}`

Do not place comments at the end of a line unless the comment is very short. In most cases, comments should be placed above the code.	`public class ArrayList`
	`{`
	`  int count; // -1 indicates uninitialized array`
	`}`

Remove TODO comments well in advance of a release.	

XML Documentation Guidance

XML documentation should provide information related to usage. It should help a programmer decide if they want to use the method. The following list provides best practice guidance for XML documentation.

- Add XML documentation with meaningful content.
- Use XML documentation to provide users and potential users with the information they need.
- Do not use XML documentation to discuss implementation details or other items not related to use.
- Do not add XML documentation for the sake of improving code coverage.
- Be aware of the methods with automatically generated XML documentation; for example, `new` and `construct`.

Labels and Text Guidance

The following list provides best practice guidance for labels and text.

- Use labels for text that will appear on the user interface.

- Put labels in double quotes.
- Do not concatenate multiple labels together.
- Use single quotes for text that will not appear in the user interface.

Database Guidance

The following list provides best practice guidance related to the database.

- Include a `try catch` around all transactions that could result in deadlock.
- Make sure the `try` for a deadlock is idempotent meaning no matter how many times the `try` is attempted, it will yield the same result.
- Consider the clarity when deciding the number of return statements in a method.
- Use `throw` instead of `ttsAbort`.
- Avoid display methods where possible.
- Set Optimistic Concurrency Control (**OccEnabled**) to **Yes** for most tables.
- Do not include user interaction inside a database transaction.
- Keep database transactions as short as possible.
- Run code on the Application Object Server (AOS) whenever possible.
- Use `where` clauses in `select` statements and in queries that align with indexes.
- If method calls are used to test conditions, put the method calls after the other conditions. If the other conditions fail, then you will not incur the cost of running the method.
- Minimize the size of database transactions.
- Consider specifying a field list in `select` statements to increase performance.
- Use `firstonly` where applicable to increase performance.
- Use aggregates in the selection criteria instead of having the code do the aggregation. If aggregations are issued in the `select` statement rather than in code, the processing is done at the database server which is much more efficient.
- Use table joins instead of nested `while` loops. Whenever possible use a `join` in the `select` statement rather than using a `while` loop and then an inner `while` loop on the related table. This reduces the amount of communication between the AOS and the database.
- Do not include any sort of user interaction in a transaction.

Exceptions Guidance

The following list provides best practice guidance related to exceptions.

- Throw an exception to stop the currently executing X++ call stack.
- Include a localized error message with all thrown exceptions.
- Use the info, warning, and error functions without a thrown exception in cases where the executing X++ call stack should not be stopped.
- Use `throw` with the static helpers on the `Error` class such as `Error::missingParameter` and `Error::wrongUseOfFunction` for errors targeted at developers.
- Do not throw an exception for an error condition that you expect will need to be caught.
- Do not throw an exception for invalid assumption cases where a `Debug::assert` is more appropriate.

See Also

Best Practice Compiler Enforced Checks
X++ Coding Standards
Naming Conventions
Designing a Microsoft Dynamics AX Application

Best Practice Compiler Enforced Checks

When using the Microsoft Dynamics AX development environment, you should adhere to a set of best practices. The X++ compiler checks the code for best practice issues. These issues can result in best practice errors, warnings, or informational messages.
In the following topics you will find summaries of the new best practices added for Microsoft Dynamics AX by release, and how to check your own code and application objects for best practices.

List of Best Practice Error and Warning Messages

When using the Microsoft Dynamics AX development environment, you should adhere to a set of best practices. The X++ compiler checks the code for best practice issues. These issues can result in best practice errors, warnings, or informational messages.
Error and warning messages require changes in the X++ code, but informational messages do not require any user action. Informational messages are rare and self-explanatory.
The following sections list error and warning messages. The usage of these messages can vary in Microsoft Dynamics AX. For example, if you cannot find a warning message in the warning message table, it might be in the error message table. Each table is sorted alphabetically by the message text. All error and warning messages are shown exactly as they appear in the code.

Error Messages

The following table lists the best practice error messages. Many error messages are also discussed in more detail in other Help topics. Where appropriate, the table contains links to specific locations in other Help topics where messages are discussed in further detail.

Error message text	Description	BPError code and label
%1 %2 not used		Method Variable Not Used, @SYS60464
%1 is an unwanted object name.	For more information, see How to: Add Rules for Objects.	Unwanted Object, @SYS85681
%1 property of %2 is not valid.	Ensure that the menu item name assigned to the form Web control is valid.	Form Web Control Unknown Menu Item Name, @SYS93552
Action menu item not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowNoActionMenuItem, @SYS108556
Approve outcome must exist and be enabled	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowApprovalOutcomesInvalid, @SYS108546
Category not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowTemplateNoCategory, @SYS108536

Error message text	Description	BPError code and label
Class name must by postfixed with %1	For more information, see Best Practices for Interfaces.	Class Name, @SYS87660
Code to handle the `InventDimId` field must be added to the Multisite Activation Wizard.	For more information, see Best Practices: Table Fields.	BPErrorTableFieldInventDimIdNotMultiSiteActivated, @SYS123160
Configuration key must be provided for a perspective.		Perspective Missing Configuration Key, @SYS94657
Configuration key not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowNoConfigKey, @SYS108553
Configuration Key with ID %1 is unknown.	Ensure that renamed configuration keys have not created a mismatch between the name referenced and the current IDs.	Configuration Key Unknown, @SYS73068
Control label is a copy of its display method label		Label Is Copy Of Display Method, @SYS60361
Control label is a copy of its field label		Report Label Is Copy Of Fields Label, @SYS57599
Control name %1 is not unique.		Form Control Name Not Unique, @SYS87713
CurrencyCodeField does not use an extended data type derived from CurrencyCode.	For more information, see Best Practices for Currency Code Fields.	Field Currency Code Field Invalid, @SYS89379
CurrencyCodeField must not be blank when CurrencyCode is set to CurrencyCodeField.	For more information, see Best Practices for Currency Code Fields.	Field Currency Code Field Empty, @SYS89329
CurrencyCodeTable must not be blank when CurrencyCode is set to CurrencyCodeField.	For more information, see Best Practices for Currency Code Fields.	Field Currency Code Table Empty, @SYS89328
CurrencyDateField %1 does not use an extended data type derived from Date.	For more information, see Best Practices for Table Field Properties.	Field Currency Date Field Invalid, @SYS97998

Error message text	Description	BPError code and label
CurrencyDateField must not be blank when CurrencyDate is set to CurrencyDateField.		Field Currency Date Field Empty, @SYS98000
CurrencyDateTable %1 does not have a relationship with this table, or no unique index exists on the target end of a relationship with that table		Field Currency Date Table Invalid, @SYS98002
CurrencyDateTable must not be blank when CurrencyDate is set to CurrencyDateField.		Field Currency Date Table Empty, @SYS97995
Current table and table %1 have Delete Actions in both directions.	For more information, see Best Practices for Delete Actions.	Table Delete Action Both Directions, @SYS74301
Data object class %1 is missing method %2.	For more information, see Best Practices: Application Integration Framework	BPErrorAIFDataObjectExtraMethod, @SYS124617
Data object class %1 has extra method %2.	For more information, see Best Practices: Application Integration Framework	BPErrorAIFDataObjectExtraMethod, @SYS124618
Delete Actions related to an unknown table with ID: %1	For more information, see the following topics: • • Best Practices for Delete Actions	Table Delete Action Unknown Table, @SYS74302
Display/Edit method must be defined using a type	For more information, see Using the display Method Modifier.	Display Edit No Extended Return Type, @SYS55403
Display menu item not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowNoDisplayMenuItem, @SYS108559
Due date provider does not reference a valid	For more information, see	BPErrorWorkflowElementDueDateProviderInvalid, @SYS108545

Error message text	Description	BPError code and label
class implementing the WorkflowDueDateProvider interface	Workflow Best Practice Checks.	
Element outcome '%1' ActionMenuItem property does not reference a valid action menu item	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementOutcomeActionMIInvalid, @SYS108549
Element outcome '%1' ActionMenuItem property not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementOutcomeNoActionMI, @SYS108547
Element outcome '%1' ActionWebMenuItem property does not reference a valid web action menu item	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementOutcomeActionWMIInvalid, @SYS108548
Element outcome '%1' EventHandler property does not reference a valid class implementing the '%2' interface	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementOutcomeEHInvalid, @SYS108551
Enum with ID %1 does not exist.	Ensure that renamed objects have not created a mismatch between the name referenced and the current IDs.	Enum Not Exist, @SYS57821
Event handler does not reference a valid class implementing the '%1' interface	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowEventHandlerInvalid, @SYS108564
Event handler not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowNoEventHandlerError, @SYS108563
Extended Data Type is set to be right justified, should be set to left justified	For more information, see Best Practices: Performance Optimizations	BPErrorEnumRightJustified, @SYS107157
Extended data types that refer to record IDs must use RefRecId or a derived extended data type.		Type Extends Rec Id, @SYS92962
Extended data types that refer to table IDs must use RefTableId or a		Type Extends Table Id, @SYS92963

Error message text	Description	BPError code and label
derived extended data type.		
Field %1 with DEL_ prefix has configuration %2 instead of SysDeletedObjects.	For more information, see Tables Best Practice Checks.	BPErrorTableFieldDelConfigKeyConflict, @SYS107044
Field group autoreport contains too few fields (%1).		Table Field Group Missing Fields, @SYS55439
Field Help is a copy of the Enum Help		Field Help Is Copy Of Enum Help, @SYS55431
Field Help is a copy of the Extended Data Type Help of the field		Field Help Is Copy Of Extended Help, @SYS55429
Field is not a member of a field group	For more information, see the following topics: • Best Practices for Fields Belong to a Field Group • Best Practices for Field Groups	Table Field Not In Field Group, @SYS55434
Field label is a copy of the Enum label		Field Label Is Copy Of Enum Help, @SYS55430
Field label is a copy of the Extended Data Type label of the field		Field Label Is Copy Of Extended Help, @SYS55428
Field must be defined using a type		Table Field Not Defined Using Type, @SYS55426
Field with ID %1 does not exist in table %2	A field in a table can be referenced by the field ID in several ways. For example, the field ID can be referenced by a relation, or by a field group. This error can be resolved by determining where the field ID is being referenced.	Form Group Field Id Unknown In Table (Also: Table Relation Unknown Extern Field, Table Relation Unknown Field, Type Field Not Exist In Table), @SYS55418
Fields using RefRecId or a derived type must have a relation defined for that		Table Field Ref Rec Id Without Relation, @SYS92956

Error message text	Description	BPError code and label
field.		
Form group (%1) and table group (%2) have different numbers of fields. Consequently, they cannot be AOS optimized.	For more information, see Best Practice Options: AOS Performance.	Form Group Control Dif Num Of Fields, @SYS68381
Form reference does not exist %1	For more information, see Best Practice Options: Reference.	Table Unknown Form Ref, @SYS55414
Help defined on a control that cannot display Help	• For more information, see Best Practices for Form Control Properties.	Help Not Defined, @SYS85234
Hierarchy provider does not reference a valid class implementing the WorkflowHierarchyProvider interface	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementHierarchyProviderInvalid, @SYS108543
Index %1 has no fields		Table Index Without Fields, @SYS87147
Index %1 is overlapped by index %2.	For more information, see Best Practices for Indexes.	Table Overlapping Index, @SYS87145
Invalid reference to workflow category	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowTemplateCategory, @SYS108537
Label %1 cannot end with a period ('.').		Label Wrong End Sign, @SYS55433
Label and Help are equal		Label And Help Are Equal, @SYS55404
Method is not referenced in X++ code or indirectly	Add a call to the unused method, or remove the method.	Method Not Used, @SYS55408
Method run on %1 and has AOSRunMode set to %2	For more information, see Application Object RunOn Property Overview.	Method Bound Wrong, @SYS85345
Missing tag '%1' in XML documentation.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationParamTagMissing, @SYS107110
Missing tag 'returns' in XML documentation.	For more information, see	BPErrorXmlDocumentationReturnsTagMissing, @SYS107110

Error message text	Description	BPError code and label
	Best Practices: XML Documentation.	
Missing tag 'summary' in XML documentation.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationSummaryTagMissing, @SYS107110
More than one tree node with this path: %1	For more information, see Best Practice Options: Unique Tree Node Names in the AOT.	AOT Path Not Unique, @SYS68375
No caching set up for the Table		Table No Caching, @SYS55412
No caption defined	For more information, see the following topics: • Reports Best Practice Checks • Best Practices for Report Properties	Caption Not Defined, @SYS60369
No Help defined		Help Not Defined, @SYS55407
No Label defined	Add a label using **Tools** and then **Developer**.	Label Not Defined, @SYS55406
No such data source %1		Form Group Control Unknown DS, @SYS68379
Not connected to a Security Key.	Ensure that the `SecurityKey` property has been set on objects that require it.	Security Key Not Connected, @SYS73076
Object has changed ID since previous release. Old ID was %1.	It is recommended that object IDs remain unchanged, especially for tables and fields. Changing an ID value can cause errors during upgrade.	Object Id Conflict, @SYS93546

Error message text	Description	BPError code and label
	For more information, see the following topics: • Best Practices for Enum Properties • Best Practices for Class Declarations	
Object has changed name since previous release. Old name was %1.		Object Name Conflict, @SYS93547
One of the properties ParticipantProvider or HierarchyProvider must be defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementNoProvider, @SYS108542
Parent class contains abstract methods. Make class abstract or implement abstract methods %1.	For more information, see Best Practice Options: Abstract.	Class Not Marked Abstract, @SYS74077
Parent Configuration Key with ID %1 is unknown.	Ensure that the configuration key value assigned to the `ParentKey` property is valid.	Configuration Parent Key Unknown, @SYS73075
Parent Security Key with ID %1 is unknown.	Ensure that the security key value assigned to the `ParentKey` property is valid.	Security Key Unknown, @SYS74743
Participant provider does not reference a valid class implementing the WorkflowParticipantProvider interface	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementParticipantProviderInvalid, @SYS108541
Primary index %1 allows duplicates.		Table Primary Index Not Unique, @SYS90099
Property %1 must contain a label ID such as @SYS4711, not %2		Help Is Text, @SYS60289
Property %1 must contain a label ID such as @SYS4711, not %2	For more information, see the following topic:	Label Is Text, @SYS60289

Error message text	Description	BPError code and label
	• Best Practices for Labels	
Reference to action menu item is invalid	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowActionMenuItemInvalid, @SYS108557
Reference to display menu item is invalid	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowDisplayMenuItemInvalid, @SYS108560
Reference to object not in version control (%1)	In the version control system, ensure that you have created all the new objects that the code depends on.	Method Refers Local Object, @SYS86883
Reference to web action menu item is invalid	For more information, see Workflow Best Practice Checks.	BPError WorkflowWebActionMenuItemInvalid, @SYS108558
Reference to web URL menu item is invalid	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowWebURLMenuItemInvalid, @SYS108561
Referenced menu does not exist	For more information, see Best Practice Options: Reference.	Menu Reference Unknown Ref Menu, @SYS55488
Relation %1 has no fields.	For more information, see Best Practices for Table Relations.	Table Relation No Fields, @SYS92955
Required access level is No access and Security Key has been set to %1.		Menu Function Access Level No Access, @SYS74738
Required element '%1' does not reference a valid workflow element	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowTemplateRequiredElementInvalid, @SYS108538
Required element '%1' does not reference same document as the workflow template	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowTemplateRequiredDocumentInvalid, @SYS108539
RunBase classes should be able to run on 'Called From' (Ensure pack and unpack are implemented	For more information, see Best Practices: Performance	BPErrorClassRunBaseMarkedOnServer, @SYS107159

Error message text	Description	BPError code and label
correctly to allow PromptProm to marshal the class across tiers)	Optimizations	
RunBase implementations must have a static description method.	For more information, see Best Practice Options: RunBase Implementation.	Class No Static Description, @SYS72474
Security Key with ID %1 is unknown	Ensure that renamed security keys refer to the correct ID, or ensure that the security key has been created.	Security Key Unknown, @SYS73073
Table %1 does not exist.		Form Property Non Standard Value, @SYS75683
Table %1 with DEL_prefix has configuration %2 instead of SysDeletedObjects.	For more information, see Tables Best Practice Checks.	BPErrorTableDelConfigKeyConflict, @SYS107042
Table %1 with SysDeletedObjects configuration key (%2) has no DEL_ prefix.	For more information, see Tables Best Practice Checks.	BPErrorTableDelPrefixConflict, BPErrorTableDelPrefixConflict, BPErrorTableIndexDelConfigKeyConflict, @SYS107043
Table fields that refer to record IDs must use RefRecId or a derived extended data type.		Table Field Uses Rec Id, @SYS92960
Table fields that refer to table IDs must use RefTableId or a derived extended data type.		Table Field Uses Table Id, @SYS92961
Table group %1 is unknown (%2)	Check whether the reference by the form group to the table group is still valid. You might need to delete the form group control, create a new field group, and then add a new form group control.	Form Group Control No Rel Table Group, @SYS73328
Table is missing Clustered Index	For more information, see	BPErrorTableNoClusteredIndex, @SYS107155

Error message text	Description	BPError code and label
	Best Practices: Performance Optimizations	
Table is missing Primary Index	For more information, see Best Practices: Performance Optimizations	BPErrorTableNoPrimaryIndex, @SYS107156
Table with ID %1 does not exist	Ensure that renamed tables have not created a mismatch between the name referenced and the current IDs.	Table Relation Unknown Extern Table (Also: Table Relation Unknown Table, Type Extern Table Unknown), @SYS55416
Tag '%1' exists more than once in XML documentation.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationDuplicated, @SYS107215
Tag '%1' has no content in XML documentation.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationParamMissing, @SYS107150
Tag '%1' in XML documentation doesn't match actual implementation.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationParamWrongName, @SYS107113
Tag '%1' in XML documentation is not supported.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationUnsupported, @SYS107111
Tag 'exception' has no content in XML documentation.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationExceptionMissing, @SYS107150
Tag 'permission' has no content in XML documentation.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationPermissionMissing, @SYS107150
Task outcomes must contain one enabled outcome of type Complete	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowTaskNoCompleteOutcome, @SYS108552

Error message text	Description	BPError code and label
The caption of the group control is a copy of its table data group label		Caption Is Copy Of Data Group Label, @SYS68389
The class will be discontinued in a later version %1. Use: %2.	For more information, see Best Practices for Use of Discontinued Functionality.	Method Discontinued In Later Vers, @SYS69514
The Client/Server setup is different from the parent class.	For more information, see Best Practices for Method Modifiers.	Class Run On, @SYS74731
The configuration key for the Table Field is a copy of the configuration key for the Base Enum.		Table Field Configuration Key Is Copy, @SYS91245
The configuration key for the Table Field is a copy of the configuration key for the Extended Data Type.		Table Field Configuration Key Is Copy, @SYS91243
The control Help text is a bad copy, and it should not be defined here.		Field Help Is Copy Of Enum Help (Also: Field Help Is Copy Of Extended Help), @SYS72533
The CurrencyCodeTable %1 does not have a relationship with this table, or no unique index exists on the target end of a relationship with that table.		Field Currency Code Table Invalid, @SYS89330
The Dimension field must always be the only field in the Dimension group.		Table Field Group Missing Fields, @SYS74735
The fields in the relation are incompatible. '%1.%2' is %3 characters too short.	Ensure that the data types of the fields are identical, or at least compatible between the two sides of the relation. Ensure that the strings in the foreign key are at least as long as the strings in the corresponding primary key.	Table Relation Fields Incompatible (Also: Type Fields Incompatible), @SYS55422

Error message text	Description	BPError code and label
The form size exceeds the maximum of %1 * %2 pixels. Current size %3 * %4 (%5\% * %6\%).		Form To High, @SYS75346
The keyword forceliterals must not be used in the query expression		TwC Dangerous API, @SYS81941
The method will be discontinued in a later version %1%2%3. Use %4	For more information, see Best Practices for Use of Discontinued Functionality.	Method Discontinued In Later Vers (Also: Method Dict Method Display Id Not Used), @SYS68910
The primary key field cannot be edited on update (AllowEdit must be set to No)		Table Primary Key Editable, @SYS60598
The primary key field must be mandatory.		Table Primary Key Not Mandatory, @SYS56378
The referenced application object does not exist (%1 %2).	For more information, see Best Practices for Existence of Referenced Application Objects.	Menu Function Unknown Ref Object, @SYS72553
Title field %1 must be declared.	For more information, see Best Practices for Declared Title Fields.	Table Title Field1 Not Declared, @SYS56377
Title field 2 must be different from title field 1	For more information, see the following topic: • Best Practices for Declared Title Fields	Table Title Field2 Not Declared, @SYS83885
TwC: Assert usage of API %1 because it is protected by Code Access Security.	For more information, see the following topics: • Code Access Security • Secured APIs	TwC Dangerous API, @SYS98156
TwC: Parameters to API %1 must be validated.	When code contains calls to system or	TwC Dangerous API, @SYS90609

Error message text	Description	BPError code and label
	kernel methods that may harm Microsoft Dynamics AX, the parameter data passed into those calls must be reviewed to ensure that the calls are harmless. After review, you may need to suppress the best practice error. For more information, see the following topic: • APIs Turned Off by Default	
TwC: Validate data displayed in form is fetched using record level security. Dangerous API %1 used.	For more information, see Best Practices: Avoiding Potential Security Issues.	BPErrorTwCEnsureRecordLevelSecurity, @SYS98155
Type Help is a copy of the Enum Help		Type Help Is Copy Of Enum Help, @SYS55451
Type label is a copy of the Enum label		Type Help Is Copy Of Enum Help, @SYS55450
Type label is a copy of the Extended (..) Data Type label of the type		Type Label Is Copy Of Extended Help, @SYS55448
Unique index %1 contains field %2 with SysDelete configuration config key assigned to it.	A field in an index has been made obsolete by an upgrade to Microsoft Dynamics AX because it was marked with `SysDelete`. This field was part of a unique index. Redesign the unique index. For more information, see Unique Indexes.	Table Sys Delete Field Index, @SYS99948
Unique index error:	For more	Table Unique Index Error, @SYS93535

Error message text	Description	BPError code and label
Fields removed from unique index: %1. Upgrade script required.	information, see .	
Unique index error: Previous nonunique index is now unique. Upgrade script required.	For more information, see Unique Indexes.	Table Unique Index Error, @SYS93534
Unique index error: Unique index introduced. Upgrade script required.	For more information, see Unique Indexes.	Table Unique Index Error, @SYS93533
Use Client/Server neutral functionality. Do not use: %1%2%3. Use: %4.		Method Neutral Funct Not Used, @SYS54379
Version mismatch of packed container. Check implementation of SysPackable interface.	For more information, see Best Practices for Interfaces.	Class Sys Packable, @SYS93536
Workflow document does not reference a valid class deriving from WorkflowDocument	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowDocumentInvalid, @SYS108555
Workflow document not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowNoDocument, @SYS108554
XML documentation is not well-formed.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationNotWellFormed, @SYS107112

Warning Messages

Warning message text	Description	BPError code and label
%1 on control is set to nonauto (Date format)		Report Date Format Set Non Auto, @SYS60296 + @SYS23272
%1 on control is set to nonauto (Decimal separator)		Report Date Format Set Non Auto, @SYS60296 + @SYS24260
%1 on control is set to nonauto (%2)		Report Thousand Sep Set Non Auto, @SYS60296
A data entry form should have at least two tab pages.	For more information, see Forms Best Practice Checks.	Form Property Non Standard Value, @SYS84385
A display or edit method has the		Table Field Has Same Name As Method, @SYS97063

Warning message text	Description	BPError code and label
same name as this field. Rename the method or the field, and check whether field groups that contain this field should contain the method instead.		
Adjustment property for field %1 of table %2 does not match its related field %3 of table %4		Table Relationship Field Adjustment, @SYS91673
A document handling button on an Action Pane should have its Name property set to "Attachments".	For more information, see Best Practices: List Pages.	BPErrorListPageDocHandlingCmdButtonNameNotAttachments, @SYS116209
A document handling button on an Action Pane should use the label @SYS114630 for its Text property.	For more information, see Best Practices: List Pages.	BPErrorListPageDocHandlingCmdButtonTextNotAttachments, @SYS116210
A List Page must have a grid.	For more information, see Best Practices: List Pages.	BPErrorListPageFormHasNoGrid, @SYS116225
A List Page must have a single Action Pane.	For more information, see Best Practices: List Pages.	BPErrorListPageFormHasNoActionPane , BPErrorListPageFormHasTooManyActionPanes, @SYS116224
All buttons on an Action Pane should have their ShowShortcut properties set to "No" to suppress the addition of extra characters for pneumonic usage.	For more information, see Best Practices: List Pages.	BPErrorListPageActionPaneButtonShowShortcutNotNo, @SYS116207
An Action Pane should not be present on a form that isn't a List Page or other Content Page.	For more information, see Best Practices: List Pages.	BPErrorFormHasActionPane, @SYS116229
AnalysisSelection		Table Analysis Selection Auto, @SYS89276

Warning message text	Description	BPError code and label
should not be Auto for a table that is visible for analysis.		
AnalysisVisibility should not be Auto for a field in a table that is visible for analysis.		Field Analysis Visibility Auto, @SYS89279
AnalysisVisibility should not be Auto for a nonsystem table.		Table Analysis Visibility Auto, @SYS89275
AnalysisVisibility should not be Auto for security keys that have no parent security key	For more information, see Best Practices for Analysis Visibility.	Security Key Analysis Visibility Auto, @SYS89711
Class should have at least one member	For more information, see Best Practice Options: Missing Member Function.	Class Missing Member, @SYS55390
Configuration Key is %1	Ensure a valid configuration key name is being used, rather than a placeholder value like "Not decided."	Configuration Key Specific (Also: Configuration Parent Key Specific, Security Key Specific), @SYS72461
Consider %1 method to run on %2 because it uses: %3	For more information, see the following topics: • Application Object RunOn Property Overview • Object Class	Method Consider Run On, @SYS54211
Consider alternative to single quoted text %1 appearing in %2	For more information, see Best Practice Options: Single Quoted Text.	Method Single Quoted Text, @SYS68040
Consider autodeclaring the form control %1		Method Not Auto Declared, @SYS68393
Consider restructuring the	For more information, see	Method Consider Restructuring, @SYS54324

Warning message text	Description	BPError code and label
%1 method because it has calls to the %2 server methods: %3, and the %4 client methods: %5.	Best Practices for Method Modifiers.	
Consider use of delete_from because method contains 'while select ... ..delete()'		Method Delete From Not Used, @SYS55398
Consider use of more specialized intrinsic functionality because method contains %1	For more information, see Intrinsic Functions.	Method Identifier Str Used, @SYS55399
Consider using a field list for select of %1. Only %2% of record size is used.		Select Using Field List, @SYS91289
Consider using keyword 'firstonly' for select of %1.		Select Using First Only, @SYS91288
Control is not defined using anything (type, field or method)	Assign a source of information to the report control. Bind the control to a type, field, or method, or remove the control.	Report Control Use Not Defined, @SYS60363
Control is set to fixed width	For more information, see Best Practices for Form Control Properties.	Report Control Set Fixed Width, @SYS60297
CurrencyCode should be SecondaryCurrency when the field uses an extended data type derived from AmountMSTSecondary and the field is visible for analysis.	For more information, see Best Practices for Currency Code Fields.	Field Currency Code Secondary Currency, @SYS89712
CurrencyDate should not be Auto		Field Currency Date Auto, @SYS98001

Warning message text	Description	BPError code and label
when a field is using an extended data type derived from money or moneyMST and the field is visible for analysis		
Display methods must be typed ('%1 %2')	For more information, see Best Practices for Unique Labels.	Table No Extended Return Type, @SYS60362
Display/edit method does not use an Enum or Extended Data Type as return type: %1	For more information, see Best Practices for Use of Labels.	Table No Extended Return Type, @SYS72489
Do not disable the control by setting Enabled to No. Set AllowEdit to No and Skip to Yes.	For more information, see Best Practices for Disabling Technique.	Form Disabling Technique, @SYS72538
Do not write to parameters (such as %1 in line %2, column %3)		Method Variable Dont Write To Parms, @SYS60115
Duplicated user interface texts. Fields: %1.	For more information, see Best Practices for Labels.	Table Duplicate UI Text Field, @SYS75650
Duplicated user interface texts. Method %1.	For more information, see Best Practices for Labels.	Table Duplicate UI Text Method, @SYS72498
Element outcome '%1' EventHandler property should be defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowElementOutcomeNoEH, @SYS108550
Enum field is Mandatory		Table Field Enum Is Mandatory, @SYS55432
Enum is not referenced in X++ code, in the table field or in an Extended Type		Enum Not Used, @SYS55470
Event handler should be defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowNoEventHandlerWarning, @SYS108562

Warning message text	Description	BPError code and label
Field is not referenced in X++ code	Add a reference to the field, or remove the unreferenced field.	Table Field Not Used, @SYS55427
FieldGroup AutoReport does not exist.		Table Missing Group Auto Report, @SYS55415
Help must end with a period or a question mark.		Help End Wrong Sign, @SYS72462
If Adjustment is set to Left, the StringSize for field %1 of table %2 must be greater than or equal to its related field %3 of table %4.	Increase the `StringSize` of the foreign key field.	Table Relationship Foreign Key To Short, @SYS91675
If Adjustment is set to Right', the StringSize for field %1 of table %2 must match that of its related field %3 of table %4.	For more information, see the following topic: • Table Field Properties	Table Relationship Field String Length, @SYS91674
Illegal name %1 %2: %3. Use parent, child, or sibling.	Terms like father, mother, sister, and brother should not be part of a member name. Replace the improper term with parent, child, or sibling.	Method Illegal Name, @SYS57827
Implement static construct to allow for modifications.	For more information, see Best Practices for Static Construct Methods.	Class No Static Construct, @SYS82256
Label is changed on the control from %1 to %2		Label Changed At Control, @SYS60298
Label on control is set to fixed width	For more information, see Form Control Properties.	Report Controls Label Set Fixed, @SYS60295
List Page Action	For more	BPErrorListPageControlVerticalSpacingNotZero,

Warning message text	Description	BPError code and label
Panes must have their VerticalSpacing property set to zero.	information, see Best Practices: List Pages.	@SYS116212
List Page Action Panes must have their Width property set to "Column width".	For more information, see Best Practices: List Pages.	BPErrorListPageActionPaneWidthNotColumnWidth, @SYS116211
List Page controls must not have any vertical spacing between them.	For more information, see Best Practices: List Pages.	BPErrorListPageControlVerticalSpacingNotZero, @SYS116208
List Page datasources must have their AllowCreate set to "No".	For more information, see Best Practices: List Pages.	BPErrorListPageFormDataSourceAllowsCreate, @SYS116227
List Page datasources must have their AllowEdit set to "No".	For more information, see Best Practices: List Pages.	BPErrorListPageFormDataSourceAllowsEdit, @SYS116226
List Page datasources must have their StartPosition set to "First".	For more information, see Best Practices: List Pages.	BPErrorListPageFormDataSourceStartPositionNotFirst, @SYS116228
List Page grids must have their AllowEdit property set to "No".	For more information, see Best Practices: List Pages.	BPErrorListPageGridAllowsEdit, @SYS116213
List Page grids must have their Datasource property set to a valid datasource.	For more information, see Best Practices: List Pages.	BPErrorListPageGridDataSourceEmpty, @SYS116217
List Page grids must have their DefaultAction property set to a button on the form. The DefaultAction property should normally point to a button that performs the "Open" action.	For more information, see Best Practices: List Pages.	BPErrorListPageGridDefaultActionEmpty, @SYS116214
List Page grids must have their	For more information, see	BPErrorListPageGridHeightNotColumnHeight, @SYS116215

Warning message text	Description	BPError code and label
Height property set to "Column height".	Best Practices: List Pages.	
List Page grids must have their ShowRowLabels property set to "Yes".	For more information, see Best Practices: List Pages.	BPErrorListPageGridShowRowLabelIsNotYes, @SYS116216
List Page grids must have their Width property set to "Column width".	For more information, see Best Practices: List Pages.	BPErrorListPageGridWidthNotColumnWidth, @SYS117724
List Pages must have a name that ends with "ListPage".	For more information, see Best Practices: List Pages.	BPErrorListPageFormNameDoesNotEndInListPage, @SYS116218
List Pages must have their BottomMargin property set to "Auto".	For more information, see Best Practices: List Pages.	BPErrorListPageFormBottomMarginNotAuto, @SYS116222
List Pages must have their LeftMargin property set to "Auto".	For more information, see Best Practices: List Pages.	BPErrorListPageFormLeftMarginNotAuto, @SYS116220
List Pages must have their RightMargin property set to "Auto".	For more information, see Best Practices: List Pages.	BPErrorListPageFormRightMarginNotAuto, @SYS116221
List Pages must have their TitleDatasource property set.	For more information, see Best Practices: List Pages.	
List Pages must have their TopMargin property set to "Auto".	For more information, see Best Practices: List Pages.	BPErrorListPageFormTopMarginNotAuto, @SYS116219
List Page Action Panes must have their VerticalSpacing property set to zero.	For more information, see Best Practices: List Pages.	BPErrorListPageControlVerticalSpacingNotZero, @SYS116212
List Page Action Panes must have their Width property set to "Column width".	For more information, see Best Practices: List Pages.	BPErrorListPageActionPaneWidthNotColumnWidth, @SYS116211
Method availability can be set explicitly		Method Access Can Be Set Explicitely, @SYS68392

Warning message text	Description	BPError code and label
to %1 via the Standard Public setting.		
Method contains code in unrequired braces %1 .... }	For more information, see X++ Layout.	Method Non Needed Block Style Used, @SYS59225
Method contains constant numeric value: %1	For more information, see X++ Standards: Constants.	Method Constant Numeric Arg Used, @SYS55396
Method contains labels in single quotes: >%1<		Method Label In Single Quotes, @SYS55395
Method contains parenthesis round case constant: %1		Method Parenthesis Round Case Const, @SYS55397
Method is empty	For more information, see Best Practices for Empty Methods.	Method Is Empty, @SYS68904
MinNoOfDecimals is greater than NoOfDecimals	For more information, see Form Control Properties.	Form Control Min No Of Decimals (Also: Report Control Min No Of Decimals), @SYS96235
Missing super call in new method of sub class.		Method Missing Super Call, @SYS62822
Module not defined	For more information, see Workflow Best Practice Checks.	BPErrorWorkflowCategoryNoModuleDefined, @SYS108540
New should be protected.	For more information, see Best Practices for new and static new... Methods.	Class New Not Protected, @SYS82255
No self relation set up for the Table. Rename function will not be available.		Table No Self Relation, @SYS56050
No unique index set up for the table	For more information, see Unique Indexes.	Table No Unique Index, @SYS60691
Only parameters must start with an underscore, not variables such as	For more information, see the following topics:	Method Variable With Underscore, @SYS60113

Warning message text	Description	BPError code and label
%1	• Best Practices for Parameters • Naming Conventions: Underscores	
Relation line %1 has possible errors in setup of the Configuration Keys. Field %2 has Configuration Key set %3 and field %4 has Configuration Key set %5.		Configuration Key Sets Not Ok, @SYS74477
Relation line %1 has possible errors in setup of the Configuration Keys. Type %2 has Configuration Key set %3 and field %4 has Configuration Key set %5.		Configuration Key Sets Not Ok, @SYS74534
Relation to table %1 (using %2) which is not in this table collection	For more information, see Best Practices for Relations.	Table Collection Relation, @SYS68398
Relations defined for a single record ID field should be defined on the extended data type for that field.		Table Field Ref Rec Id Relation On Table, @SYS92957
Report design orientation is not set to Auto	For more information, see Best Practices for Report Properties.	Report Des Orientation Not Set Auto, @SYS60368
Report has generated design %1	For more information, see Best Practices for Report Design.	Report Has Generated Design, @SYS60365
Report template %1 does not exist	For more information, see Best Practices for Report	Report Unknown Template, @SYS60367

Warning message text	Description	BPError code and label
	Properties.	
Security Key is %1	Ensure a valid security key name is being used, rather than a placeholder value like "Not decided."	Security Key Specific, @SYS73072
Security key should not be specified on container controls because it prevents personalization.	For information about the personalization choices that are possible for container controls, see Form Control Properties.	Security Key Not Allowed, @SYS91028
SingularLabel should be provided for a table that is visible for analysis.		Table Singular Label Empty, @SYS89278
Table fields with AnalysisVisibility set to DefaultField or High should be in included in at least one perspective		Field Visible But Not In Perspective, @SYS94645
Table group is %1.		Table No Table Group, @SYS55413
Table has a record ID index but does not seem to use the record ID field explicitly for lookup		Table Rec Id Field Used Useless, @SYS60597
Table has no record ID index but does use the record ID field explicitly in relation in %1		Table No Record Id Index, @SYS60524
Table has no record ID index but does use the record ID field explicitly in select ... where in %1		Table No Record Id Index Select, @SYS60523
Table has no record ID index but uses it %1 times		Table No Record Id Index But Used, @SYS60522
Table has record ID		Table Record Id Index Not Use Field, @SYS60520

Warning message text	Description	BPError code and label
index but does not use record ID field explicitly		
Table is using CreatedDateTime +or ModifiedDateTime, RecId index needs to be created.	For more information, see Tables Best Practice Checks.	BPErrorRecIDNeededCreatedModifiedDateTime, @SYS127410
Tables with AnalysisVisibility set to High, Medium, or Low should be included in at least one perspective	For more information, see Best Practices for Analysis Visibility.	Table Visible But Not In Perspective, @SYS94641
Tables with only one index should have it defined as a cluster index	For more information, see Clustered Indexes.	Table One Index Not Cluster, @SYS68395
Tag '%1' in XML documentation is not supported.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationUnsupported, @SYS107111
The Construct method must only instantiate the class. Consider using a static new pattern instead.		Class Construct Pattern, @SYS82257
The `CurrencyCode` property should not be **Auto** if the field is derived from the money Extended Data Type and the AnalysisVisibility property is set to High or Low.		Field Currency Code Auto, @SYS89378
The designs property %1 is disabled and data source %2 has property %3 on table %4 set to true. Set the designs property %1 to Yes to ensure that the	Set the design property to Yes.	Form Property Non Standard Value, @SYS77537

Warning message text	Description	BPError code and label
form restarts in the previous company.		
The design property %1 is enabled, but the property %3 on table %4 for data source %2 has not been set correctly. Set the design property %1 to No to prevent the form from restarting in the previous company.	Set the design property to No.	Form Property Non Standard Value, @SYS77486
The group could be based on a (new) table field group		Form Group Could Be Based On New Group, @SYS68387
The group is empty		Form Group Is Empty, @SYS68388
The group should be given a logical name	For more information about groups on forms, see Forms Best Practice Checks.	Form Group No Logical Name, @SYS68385 = @SYS68384
The primary index should be defined because a unique index exists	For more information, see the following topic: • Best Practice Options: Use of Indexes	Table Unique Index No Primary, @SYS68396
The property %1 has a nondefault value %2. Expected %3.	All form properties that have an Auto or Default setting should be kept at that setting. For more information, see Best Practices for Form Design Properties.	Form Property Non Standard Value, @SYS72374
The property %1 should be set to %2.		Form Property Non Standard Value, @SYS84109
The word %1 is not spelled correctly.	For more information, see Best Practices for Spelling.	Doc Node Spelling Mistake, @SYS84009

Warning message text	Description	BPError code and label
This class without members does not extend any other class	For more information, see Best Practice Options: Missing Member Function.	Class No Member Not Extend, @SYS55391
This date construction can be illegal: %1	For more information, see Best Practice Options: Date Features.	Method Illegal Date Construction, @SYS68391
ttsbegin/ttscommit are unbalanced with %1	For more information, see X++ Standards: ttsBegin and ttsCommit.	Method Unbalanced Ttsbegin Commit, @SYS57826
TypicalRowCount should not be Auto for a table that is visible for analysis.		Table Typical Row Count Auto, @SYS89277
Unextended class without members is not extended by any other class	Add a member to the class, or remove the class.	Class Unextended Not Extend, @SYS55392
Use other construction than this illegal one: %1%2%3	For more information, see Best Practices for Static Construct Methods.	Method Illegal Construction Used, @SYS55400
Variable %1 is not written, but read.	For more information, see Best Practices for Use of Variables.	Method Variable Read But Not Written, @SYS60114
Wrong security key. Security key must match position in Main Menu.		BPCheck, @SYS76678
XML documentation not written for this method.	For more information, see Best Practices: XML Documentation.	BPErrorXmlDocumentationMissing, @SYS107198

See Also

Setting Up Best Practices Checks

Best Practices: Avoiding Potential Security Issues

New Best Practices in Microsoft Dynamics AX 2009

When using the Microsoft Dynamics AX development environment, you should adhere to a set of best practices. The X++ compiler checks the code for best practice issues. These issues can result in a best practice message that is an error, warning, or informational. Best practice checks are recommended for any Microsoft Dynamics AX partner or end user who is enhancing or customizing Microsoft Dynamics AX. The following sections provide information about the best practice checks that were added for this release. For information about Microsoft Dynamics AX 4.0, see New Best Practices in Microsoft Dynamics AX 4.0.

Application Integration Framework

Best practice checks for Application Integration Framework (AIF) verify that the data object is synchronized with the underlying artifacts that were used to define the data object. The best practice check is for missing or extra methods in the data object. For more information, see Best Practices: Application Integration Framework.

List Pages

The goal of the best practice checks for list pages is to establish a common look and behavior for all list pages. The checks are run for new or modified list pages. For more information, see Best Practices: List Pages.

Performance

Best practice checks for performance were added regarding indexes, extended data types, minimizing remote procedure calls, and details of `RunBase` classes. For more information, see Best Practices: Performance Optimizations.

Security

A best practice check for security was added to alert you when a dangerous API was used and a review is required. For more information, see Best Practices: Avoiding Potential Security Issues.

Tables

Best practice checks were added to prevent problems that occur when objects are marked to be deleted but are not linked to a `SysDeletedObjects` configuration key and vice versa. For more information, see Tables Best Practice Checks.

Table Fields

A best practice check was added to verify that when an `InventDimId` field is added to a table, code for handling the field is also added to the Multisite Activation Wizard. For more information, see Best Practices: Table Fields.

Workflow

Best practice checks for workflow focus on the properties of workflow categories, templates, approvals, and tasks. The checks verify that the properties are correctly configured for execution. For more information, see Workflow Best Practice Checks.

XML Documentation

Best practice checks for XML documentation focus on the validity of the documentation provided. The checks verify that particular sections are included. For more information, see Best Practices: XML Documentation.

See Also
Best Practices for Microsoft Dynamics AX Development
List of Best Practice Error and Warning Messages
New Best Practices in Microsoft Dynamics AX 4.0

New Best Practices in Microsoft Dynamics AX 4.0

When using the Microsoft Dynamics AX development environment, you should adhere to a set of best practices. The X++ compiler checks the code for best practice issues. These issues can result in a best practice message that is an error, warning, or informational. Best practice checks are recommended for any Microsoft Dynamics AX partner or end user who is enhancing or customizing Microsoft Dynamics AX.

Disabling Best Practices Warnings

You can disable individual warnings caused by a best practices deviation in a particular piece of X++ code. For more information, see Setting Up Best Practices Checks.

Classes

It is a best practice to have a static construct method for every X++ class. For more information, see Best Practices for Static Construct Methods.
The instance `new` method should be protected. For more information, see Best Practices for new and static new... Methods.

Form Design

The maximum size for a form has been increased from 800 x 500 to 824 x 668. For more information, see Forms Best Practice Checks.
The first two tab pages of data entry forms must be **Overview** and **General**.
New best practices for form design properties are:

- `HideToolbar` should be set to **No** for data entry forms and **Yes** for lookup forms.
- `Columns` should be set to 1 for data-entry forms.
- `TitleDatasource` should be the same as the data source if there is only one data source.
- `Caption` should be the same as the label defined in the `TitleDatasource` property for the table if there is only one data source.

For more information, see Best Practices for Form Design Properties.

Table Fields and Extended Data Types

Do not use the system `recID` or `tableID` data types directly for table fields, or for extended data types. Instead, use `RefRecID` or `RefTableID`.
If a table field uses `RefRecID` or a type derived from it, a relationship must be defined for that field.
Do not create a table relationship for a single field if the field is derived from `RefRecId`. Create a relationship on the extended data type for that field instead.
For more information, see Best Practices: Table Fields, Extended Data Types Best Practice Checks, and Best Practices for Table Relations.

Code Layout

You should now put braces round every code block—even if the block contains only a single line. Previously, it was acceptable not to put braces around single-line code blocks as long as they did not form part of an `if...else` statement.
Lines of code must start in tab positions (1, 5, 9, and so on). It is an error if a line of code starts in columns 2, 3, or 4.

Certain reserved words, such as `if`, `else`, `switch`, and `while` must be at the start of a code line and in a tab position.
For more information, see X++ Layout.

Comments

Use only the "//" comment notation.
Do not put comments at the end of a code line. Put them on the line before the relevant line of code.
For more information, see X++ Standards: Comments.

Naming conventions

Do not begin a name with "aaa," or "CopyOf." Do not begin a name with "DEL_" unless it is a table, extended data type, or enum, and is needed for data upgrade purposes. For more information, see Naming Conventions.

Interfaces

There are new Best Practices for `SysPackable` and `SysUnitTestable`. For more information, see Best Practices for Interfaces.

See Also
Best Practices for Microsoft Dynamics AX Development

Setting Up Best Practices Checks

Check your code and application objects for compliance with best practice rules for Microsoft Dynamics AX.

Using the Best Practice Tool

To access the best practice tool

1. From the Microsoft Dynamics AX menu, point to **Tools** and then click **Options**.
2. Click the **Best Practices**.
3. In the **Warning level** list, select the checks that you want to perform.
4. Click the **OK** button.

Best practice compliance will be checked when you compile or check in an object. For information about enabling best practice checks, see How to: Enable Best Practice Checks. You can also check best practice compliance for one or more nodes in the Application Object Tree (AOT).

To check best practices for nodes in the AOT

1. In the AOT, right-click a node, point to **Add-Ins** and then click **Check Best Practices**.

Understanding the Results

Results are displayed in an **Infolog**. Descriptions of the severity codes are listed in the following table.

Severity code	Description
Informational message	Supporting information and no action is required (shown in blue).
Warning message	Violations that you should consider fixing

Severity code	Description
	(shown in yellow).
Error message	Severe violations that must be fixed (shown in red).

If you double-click a message, the code editor will open at the offending line of code.

Fixing the Violations

Fix the violations for as many of the warnings and errors as possible. The goal is to increase the quality of the code not just to reduce the number of messages in the **Infolog**.

Note:
Code that causes a best practice error can be prevented from being checked into the version control system based on version control settings.

See Also

Best Practices for Microsoft Dynamics AX Development

How to: Enable Best Practice Checks

You should check your code and application objects for compliance with the best practices for Microsoft Dynamics AX.

Enabling Best Practice Checks

Use the following procedures to enable, identify, and check your code for best practice compliance.

To enable best practice checks

1. From the Microsoft Dynamics AX menu, point to **Tools**, and then click **Options**.
2. In the **Options** form, click the **Compiler** button.
3. Set the **Diagnostic level** to **Level 4**.
4. Click **OK**.

To identify which best practice checks to enable

1. In the **Options** form, click the **Best Practices** button.
2. In the **Best Practice parameters** form, select the best practice parameter checks that you want to verify in your code.
3. Set the Warning level drop-down to **All**.

To check your code and application objects for best practice compliance

1. In the **AOT**, right-click the **element you want to verify for best practice compliance and then click Compile**. Best practice information is provided in the **Compiler output** in the **Best Practices** tab.

See Also

Setting Up Best Practices Checks
How to: Suppress Best Practice Checks
How to: Create Best Practice Checks
Best Practice Options
Best Practice Compiler Enforced Checks

How to: Suppress Best Practice Checks

Most best practice violations are intended to be resolved but you may have a valid scenario where you want to suppress the violation. Violations can be suppressed at design time or run time.

Suppressing Best Practice Violations

In this section, you will suppress best practice violations at design-time. You will mark the violation and the best practice tool will reduce the severity of the violation error or warning to a level of an informational message.

To suppress a best practice check in code

1. In code editor, add the following code before the offending best practice violation.

```
//BP Deviation Documented
```

To suppress a best practice violation in the model or at run-time, you must know the error code and error code text.

To suppress a best practice check in the model

1. In the Compiler output, on the **Best Practices** tab, right-click the best practice violation you want to suppress, and then click **Record Info**.
2. In the Record information dialog box, click the **Show all fields** button. The error code is found in the **Other fields** section of the Compiler information window.
3. In the AOT, expand the **Macros** node and double-click **SysBpCheck**.
4. Press CTRL+F to find the error code you identified in step 2.In the AOT, expand the **Macros** node and double-click **SysBpCheckIgnore**.
5. Add a line that contains the AOT path and the error code text to ignore. Your code to suppress a best practice violation will resemble the following.

```
<violation errorcode="#BPErrorObjectNameConflict"
    path="\Classes\EmplADImport"/>
```

In this section, you will suppress best practice violations at run-time. Suppressing violations in this manner sets the best practice tool to not acknowledge certain types of violations. You must know the error code text as described earlier in this topic.

To suppress best practice check at run time

1. In the AOT, right-click **Jobs**, and then click **New Job**.
2. Add the following code.

```
SysBpCheck::ignore(#error_code_text_that_you_want_to_suppress);
```

3. Press F5 to run the job.

The best practice error will be suppressed until the Microsoft Dynamics AX session ends.

See Also

Setting Up Best Practices Checks
How to: Enable Best Practice Checks
How to: Create Best Practice Checks
Best Practice Options
Best Practice Compiler Enforced Checks

How to: Create Best Practice Checks

The best practice check tool is implemented in X++ so that you can customize or extend the best practice tool.

Creating Best Practice Rules

There is a best practice class for each element in the AOT.

To find the best practice class

1. In the AOT, expand the Classes node and find the best practice class for the element you want to create a best practice rule for. The class name will start with `SysBPCheck` followed by the AOT element. For example, the `SysBPCheckTable` class contains best practice rules for tables.

2. Right-click the class and click **New Method**. You will include your code for the best practice check in the new method.

For example, this best practice rule will check that developers are using comment headers. This method was added to the `SysBPCheckMemberFunction` class.

```
private void checkCommentHeader()
{
    str source = memberFunction.AOTgetSource();
    ;
    // Check if the comment is included.
    if (!strStartsWith(source, '//'))
    {
        // Report the error.
        sysBPCheck.addError(10000, 1, 1, "Missing header comment");
    }
}
```

You must also add a line to the `check` method of the class to call your new best practice check method.

```
this.checkCommentHeader();
```

Note:

When you compile a method without the header information, the Compiler output dialog box will report the best practice error.

See Also

Setting Up Best Practices Checks
How to: Enable Best Practice Checks
How to: Suppress Best Practice Checks
Best Practice Options
Best Practice Compiler Enforced Checks
Best Practices for Microsoft Dynamics AX Development

Best Practice Options

Use the **Best Practice Options** form to select which best practice checks to verify. Best practice checks help to make sure that the best practice guidelines are followed.
Select the **Warning level** to specify the kinds of messages produced. The warning levels are as follows:

- **Errors only** - Messages only about best practice errors
- **Errors and warnings** - Messages only about best practice errors and warnings

- **All** - Messages about violations to all best practices checks

Use the check box to mark or clear individual checks.

Note:
Disabling a check suppresses the messages about violations to the check, not the check itself.
To see descriptions of the best practices checks, click the names of the checks that appear in the tree in the right pane of the **Best Practice parameters** dialog.

See Also
Best Practices for Microsoft Dynamics AX Development

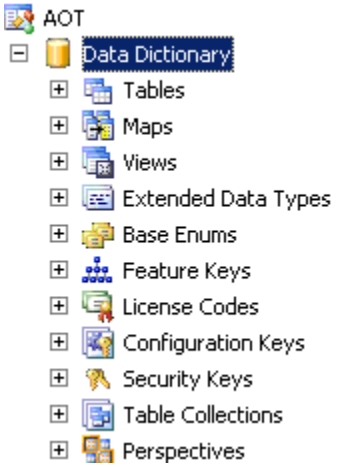
Best Practice Compiler Checks for Application Objects

This section of the SDK describes the best practices for objects in the application object tree (AOT). It includes advice about how to set object properties.

- Data Dictionary Best Practice Checks (includes best practices for tables, views, extended data types, base enums, configuration keys, security keys, table collections, and perspectives).
- Macros Best Practice Checks
- Classes and Methods Best Practice Checks
- Forms Best Practice Checks
- Reports Best Practice Checks
- Queries Best Practice Checks
- Jobs Best Practice Checks
- Menu Items Best Practice Checks
- Web Best Practice Checks

Data Dictionary Best Practice Checks

The Data Dictionary node in the AOT contains the following items:



This section of the SDK contains programming standards related to the following data dictionary items:

- Tables, including Fields, Field Groups, index design and index properties, relations and table methods
- Views
- Extended Data Types
- Base Enums
- Configuration Keys and Security Keys

- Table Collections
- Perspectives

See Also
Best Practice Compiler Checks for Application Objects

Tables Best Practice Checks

This section of the SDK describes the best practices for tables.

Best Practice Checks
The following table lists the best practices error messages and how to fix the errors.

Message	Message type	How to fix the error or warning
Table %1 with SysDeletedObjects configuration key (%2) has no DEL_prefix.	Error	The object is linked to a SysDeletedObjects configuration key but is not marked as DEL_. Mark the object as DEL_.
Table %1 with DEL_prefix has configuration %2 instead of SysDeletedObjects.	Error	The object is marked as DEL_ but it is not linked to a SysDeletedObjects configuration key. Unmark the object as DEL_ or link to the SysDeletedObjects configuration key.
Table is using CreatedDateTime +or ModifiedDateTime, RecId index needs to be created.	Error	The RecId-index must be enabled when either the CreatedDateTime or ModifiedDateTime is enabled on the table.
Index %1 with SysDeletedObjects configuration key (%2) has no DEL_prefix.	Error	The object is linked to a SysDeletedObjects configuration key but is not marked as DEL_. Mark the object as DEL_.
Field %1 with DEL_ prefix has configuration %2 instead of SysDeletedObjects.	Error	The object is marked as DEL_ but it is not linked to a SysDeletedObjects configuration key. Link to the SysDeletedObjects configuration key.
Field %1 with SysDeletedObjects configuration key (%2) has no DEL_ prefix.	Error	The object is linked to a SysDeletedObjects configuration key but is not marked as DEL_. Mark the object as DEL_.

Best Practices for Table Properties

The following table describes the Microsoft Dynamics AX standards and best practices (rules) for setting table properties. (The properties are in the same order as the UI; related properties are grouped together.)
For a description of each property, see Table Properties.

Property	Rules
ID	Always ship a table with the same ID that it has previously shipped with. If you try to create a new table with an ID that has already been used for a table in a previous version of Microsoft Dynamics AX, an error occurs.
Name	Prefix with the Module name. For example, Cust, Vend, or Invent. Infix with a logical description of the contents. For example, CustCollectionLetterTrans, where CollectionLetter is the infix. For temporary tables, infix with Tmp. For example, CustTmpLedger. Postfix with the type of contents. For example, Trans, Jour, Line, Table, Group, Parameters, or Setup. The primary tables for the major business areas are postfixed with Table. For example, CustTable, InventTable, and VendTable.
Label	Mandatory unless the table has the MaxAccessMode property set to NoAccess. An error occurs if: • You do not set this property. –and– • MaxAccessMode is not set to NoAccess. The text value of the label must be unique across all tables (excluding temporary tables), views, and maps, in all languages.
FormRef	For tables where the TableGroup property has been set to Group, Main, or WorksheetHeader, you must do the following: • Have a form to maintain the table records. • Set the FormRef property to the name of a display menu item. The form and the display menu item that are used to start the form should have the same name as the table. An example

Property	Rules
	of this rule is the CustTable table in Microsoft Dynamics AX. **Note:** Don't set the `FormRef` property for tables where the `MaxAccessMode` property is set to NoAccess. These tables are not used in a form.
`TitleField1, TitleField2`	Mandatory unless: • The `MaxAccessMode` property for the table is set to NoAccess. –or– • The `TableGroup` property is set to Parameter. Don't set this property if there are not enough fields (according to your needs) in the table. (Fields of type `real` or `integer` should not be used as title fields.) Specify the two best fields for `TitleField1` and `TitleField2` according to the following: • `TitleField1` – The key field for the records in the table. Use a descriptive title if the key field has information for the user. • `TitleField2` – A descriptive field for the records in the table. For example, for the Table InventItemGroup table, TitleField1 is ItemGroupId, and TitleField2 is Name. If the value of `TitleField1` and `TitleField2` is the same, an error occurs.
`Temporary`	Use the `setTmp` table method to make a non-temporary table temporary rather than creating a copy of the table, and then making it temporary. The two versions of the table can quickly become out of sync.
`TableContents`	Leave this property set to Not Specified for most tables. Set to Default Data for customer-independent data. For example, time intervals and unit conversions. Set to Base Data for customer-dependent data. This data is often from an existing system that has been imported or entered into Microsoft Dynamics AX. For example, customers and vendors. Set to Default+Base Data for data that can be customer-dependent in some countries/regions, but not in others.
`SystemTable`	Set this property to Yes if you want the

Property	Rules
	table to be designated as a system table. For example, the information about the system table is used during export and import when system tables can be filtered out. Don't overuse this feature.
`ConfigurationKey`	Set the `ConfigurationKey` property for most tables. This ensures that the table is removed when the key is disabled.
`SecurityKey`	Mandatory.  This property is mandatory unless the table's `Systemtable` or `Temporary` property is set to Yes. Security keys generally follow the layout of the Navigation Pane. For example, AdminTables.
`MaxAccessMode`	Mandatory. If the table is a transactions table, set this property to View.
`CacheLookup`	Set to the EntireTable value for the following tables: • Contains static data (for example: some Main, and most Group and Parameter tables). • Are frequently accessed by the Not Found cache when it hits `select` statements. For example: • `while select`. • `==` selects something that doesn't match the cache candidate key. • `select` using relational operators other than `==` (such as `<`, `>` and so on). • Has a moderate number of records (hundreds or thousands, but not millions). • Is obvious that one `select` statement to the database outperforms `select` statements to the database. If a table performs poorly when the cache type is set to EntireTable, it is possible to change the cache setting to FoundAndEmpty or Found for a particular installation. Tables with the cache type set to EntireTable should have a Cluster index. This ensures that the table loads as quickly as possible. If the cache type is set to Found, check that records are actually found in the cache. Ensure that you are using the `find` method. Ensure that you do not over-

Property	Rules
	qualify the cache candidate key—it will not utilize the cache. Remove Found caching from tables that have no unique index.
CreateRecIdIndex	Set this property to Yes only if you actually need an index on the Record ID field. All tables have a Record ID index, but the index is set to Passive when the CreateRecIdIndex property is set to No.
SaveDataPerCompany	Set to Yes for company-specific tables. Set to No if the data is related to cross-companies, installation, a database, the AOT, tracing, or OLAP. For example, SysTraceTable or OLAPServerTable. **Note:** If the SaveDataPerCompany property on a table is set to Yes, the SetCompany property on a form design that uses that table as a data source must also be set to Yes.
TableGroup	Mandatory. Set to Group for tables that contain grouping and categorizing information. If the parent table is also a Group table, you only sometimes establish delete actions for a table that relates to a group table. Deleting records from a group table can sometimes result in an unstable situation. Enable confirm deletion. For more information, see Maintaining Data Integrity. Typical examples of Group tables from the standard application are CustGroup and VendGroup. Set to Main for tables that contain base data. **Note:** Consider using an alias field for all tables that have the TableGroup property set to Main. Alias fields are set by using the AliasFor property on the field. For example, a phone number could be an alias for a customer ID. When the phone number is entered, it is automatically replaced by the customer's ID.
PrimaryIndex	Mandatory property for tables that have a unique index.

Property	Rules
	If there is more than one unique index, this property determines which index Found caching works on.
ClusterIndex	Set the index that the table should be organized by. Leave the index blank if performance tests (on realistic data) show that clustering does not work better (CRUD, time, space). For more information, see Clustered Indexes.
AnalysisVisibility	Mandatory unless: • The Systemtable or Temporary properties are set to Yes. –or– • The MaxAccessMode is set to None. Set the AnalysisVisibility property to the following values: • High – tables that are most commonly needed on reports. For example, CustTable and VendTable. Don't use High for more than eight tables in a module. • Medium – tables that are often needed on reports. • Low – tables that are unlikely to be used for reporting. For example, Parameter tables. • None – tables that should not be shown for end-user reporting. If you set the AnalysisVisibility property to High, Medium, or Low, you must include the table in at least one perspective in the Application Object Tree (AOT).
AnalysisSelection	Mandatory when: • The AnalysisVisibility property has been set. –and– • The TypicalRowCount property has not been set.
TypicalRowCount	Mandatory when: • The AnalysisVisibility property has been set. –and– • The AnalysisSelection property has not been set. Select a value that corresponds to the number of rows that the table will probably have.
IsLookup	Set to Yes if the table consists of only a primary key and one other field.
SingularLabel	Mandatory if you have set the

Property	Rules
	AnalysisVisibility property to Yes.
ModifiedDate	Set this property to Yes only if you need the information it provides.
ModifiedTime	Set this property to Yes only if you need the information it provides.
ModifiedBy	Set this property to Yes only if you need the information it provides.
ModifiedTransactionId	Set this property to Yes only if you need the information it provides. Strongly consider setting worksheet lines to Yes. This enables you to see what transaction a change was part of. This information is useful for audits.
CreatedTransactionId	Enable the CreatedTransactionId property if your table has the TableGroup property set to Transaction. If the TableGroup property is not set to Transaction, set the CreatedTransactionId property only if you need to use information about which transaction created each record in the table.
CreatedBy, CreatedDate, CreatedTime, ChangedBy, ChangedDate, ChangedTime, LockedBy	Read-only properties.

Modified/Created Fields
The Modified/Created fields are available on all tables, but can be disabled so that space is not used for unnecessary information.
Information is also available about transaction changes for tables by doing the following:

- Enable the TransactionLog system on all transaction tables.
- Use the database log.

See Also
Tables Best Practice Checks

Best Practices: Table Fields

Microsoft Dynamics AX conducts a best practice check for table fields. For information about how to set the options for best practice checks, see Best Practice Options. For information about field properties and field groups, see Best Practices for Table Field Properties and Best Practices for Field Groups.

Best Practice Checks
The following table lists the best practice error messages and how to fix the errors.

Message	Message type	How to fix the error or warning
Code to handle the InventDimId field must be added to the Multisite Activation Wizard.	Error	You have added an InventDimId field to a table and you must also add code for handling this field to the Multisite Activation Wizard. This is necessary so that during activation, the site dimension is properly populated wherever inventory dimensions are used. You must add the InventDimId field to

Message	Message type	How to fix the error or warning
		InventSiteActivateDimFieldsCheck.updateableFields or InventSiteActivateDimFieldsCheck.notUpdatableFields. Adding it to one of these would affect whether it is considered during multisite activation or not. This error message is being sent through InventSiteActivateDimFieldsCheck.ValidateField that receives as a parameter the specific field being evaluated, you can use a breakpoint to determine which field is causing the error. The Multisite Activation Wizard activates the multisite functionality for a Microsoft Dynamics AX company. Data references by all InventDimId fields are updated with site information. Adding an InventDimId field without updating the Multisite Activation Wizard makes it impossible to activate multisite functionality.

Table IDs and Record IDs

Do not directly use the system data types recId or tableId. Instead, use the following extended data types:

- RefRecId for fields that refer to record IDs
- RefTableId for fields that refer to table ID
- Extended data types derived from RefRecId and RefTableId

If a table field uses RefRecID or a data type that is derived from it, a relationship must be defined for that field on either the extended data type or the table. . A relationship is not required, but is recommended, if the field is in a temporary table.

Memo and Container Fields

Review the use of memo and container fields in application tables. Memo and container fields have the following characteristics:

- Add time to the application fetch
- Add time to the database fetch
- Inhibit array fetching
- Cannot be used in where expressions

Consider using text fields with a specific number of characters instead of memo fields. For example, use the Addressing extended data type for addressing fields. Use the ItemFreeTxt extended data type for item descriptions on order lines, and so on.

When selecting tables where the memo or container fields are not needed, consider using a field list, which excludes the unneeded memo or container fields.

Note:

Use field lists carefully. Ensure that they are not used where additional information might be needed.

If the memo or container field is rarely used (compared with the other fields in the table), you could move them away from the table. Place them in a separate table that is then selected (joined) only when the memo and container fields are actually needed. This helps avoid potential field list problems.

See Also

Best Practices for Table Field Properties
HelpText Guidelines

Best Practices for Field Groups

Best Practices for Table Field Properties

The following table lists best practices for setting Table Field Properties.

Property	Rules
`ID`	Always ship a table field with the same ID as it has been shipped with before. If you try to create a new field with an ID that has already been used by another field in the previous version of Microsoft Dynamics AX, you will get an error.
`Type`	Always ship a table field with the same base type as it has been shipped with before. The size of the field must be the same, or greater than it has been before. Each field must be defined using a type, or you will get an error.
`Name`	A field should have the same name as the extended data type used, or it should have a logical name. The field making up the key should be post-fixed "Id," for example, "ItemId." You can remove the prefix if the name makes sense. For example CustTable.CustName could be CustTable.Name. But do not remove the prefix for ID fields (CustTable.CustId). If you try to create a field with a name that has already been used in the previous version of Microsoft Dynamics AX, you will get an error. A field cannot have the same name as an edit or display method on the same table.
`Label`	Mandatory property. It is preferable to put labels on extended data types. If you choose to overwrite the value inherited from the extended data type, the value must be different from the one for the extended data type. Do not set the `Label` property to the same property as the `HelpText` property. The label must be unique for the table. The label text cannot end with a period.
`HelpText`	Mandatory property. It is preferable to put `HelpText` on the field's extended data type. If you choose to overwrite the value, the value must be different than the one for the extended

Property	Rules
	data type or enumeration. HelpText must be a complete sentence and have ending punctuation (".", "?" or "!"). Do not set the `HelpText` property to the same property as the `Label` property; the user should be provided with more detailed information than is available in the label. For more information, see HelpText Guidelines.
`GroupPrompt`	It is a best practice to leave this property blank. If you do use this property, you must use a label.
`SaveContents`	Because virtual fields are rarely used, this property should usually be set to Yes. Instead of virtual fields, you can use Using the display Method Modifier.
`Mandatory`	If the field is the primary key (or part of the key), the property must be set to **Yes**. You should set **Mandatory = No** for enum fields. The following case is an exception: If the enum first outcome (**value = zero**) is named **None** and has the label **Not selected**. If **Mandatory = Yes** on an enum field, the `validateWrite` method will fail if the enum field has the value **zero**.
`AllowEditOnCreate`	Usually set to Yes.
`AllowEdit`	If the field is the primary key (or part of the key), the property must be set to No.
`Visible`	Usually set to Yes, but it can be set to No for system-only information (information that will not be shown on the user interface), making the field accessible only from the code.
`ConfigurationKey`	This property must have a value to disable the field. It is preferable to put a Configuration key on the field's extended data type or enum. If you choose to overwrite the value, the value you give here must be different than the one for the extended data type or enum. Disabling a table's configuration key will also disable the fields in the table.
`AliasFor`	If the user might think of other fields as keys for the table, in addition to the one

Property	Rules
	you have designated as the key, set them as alias fields. For example, on the item table, the bar code is an alias for the item ID.
AnalysisVisibility	Mandatory property if the AnalysisVisibility property has been set for the table, unless the field property Visible has been set to No.  Set to DefaultField for fields that provide the most important information about a table. Set to High for fields that are often used for reporting. Set to Low for fields that are unlikely to be used for reporting. Set to None for fields that should not be shown for end-user reporting. Try to limit the number of fields that are set to DefaultField or High to no more than 15. If you have set AnalysisVisibility to DefaultField or High, the field must be included in at least one perspective. 
ExtendedDataType	Mandatory property.  All fields must be defined by using an extended data type or an enum. Fields that contain the same information must share the same extended data type or enumeration.
Other properties	Other properties for fields depend on the data type. For integers and reals, it is a best practice to set FieldUpdate to Absolute. For strings, StringSize and Adjustment should be set on the extended data type. For reals, the CurrencyCode property must be set if the data type extends the money system type and the AnalysisVisibility property has been set  (for other reals, the property is unavailable). If the data type is derived from AmountMSTSecondary, you must set the CurrencyCode property to SecondaryCurrency. Otherwise, set it to CurrencyCodeField. If CurrencyCode is set to CurrencyCodeField, you must also set the CurrencyCodeTable property to the table containing that field, or to a table for which a relationship exists from the table containing this field, and in which a unique index exists for the fields on the target end of the relationship. If

Property	Rules
	`CurrencyCode` is set to CurrencyCodeField, you must also set the `CurrencyCodeField` property to a field by using an extended data type derived from `CurrencyCode`, in the CurrencyCodeTable. **Note:** Do not use master currency amounts to set the `CurrencyCode` properties, because master currency amounts are always in the master currency for the associated company. For reals, the `CurrencyDate` property must be set if the data type extends the `money` or `moneyMST` system types (for other reals, the property is unavailable). This property must be set to CurrentRate or CurrencyDateField if the `AnalysisVisibility` property for the field is set to DefaultField, High, or Low. If the `CurrencyDate` property is set to CurrencyDateField: • The `CurrencyDateTable` property must be set to the table that contains that field or a related table in which a unique index exists for the fields on the target end of the relationship. • The `CurrencyDateField` property must be set to a field by using an extended data type derived from `Date`.

Best Practices for Field Groups

Any field that appears in the user interface must belong to a field group. You must always use field groups to design your forms.
Standardized group names are as follows:

- Identification
- Administration
- Address
- *ModuleName* (for example 'Ledger')
- Setup
- Dimension
- Misc

The Dimension field must always be the only field in a group named 'Dimension.'

Properties

Property	Rules
`Name`	Mandatory
`Label`	Mandatory

AutoReport
You must place at least two fields in the **AutoReport** field group for each table, except for parameter tables.
The fields placed there should be the ones that the user expects to print when they first click **Print** on the **File** menu. The fields used for the `TitleField1` and `TitleField2` properties are often candidates for this field group.

AutoLookup
If you do not put anything into the **AutoLookup** field group, the AutoLookup form will contain the fields used in the `TitleField1` and `TitleField2` properties, plus the field in the first unique index.

Extended Field IDs

Extended field IDs are used to refer to a particular field within a field array. An extended field ID's array index is packed in the high word, and the actual field ID is packed in the low word as shown in the following figure.

Storage of an array index and field ID in an extended field ID
The following methods are available in the Global Class to manipulate extended field IDs:

- `fieldExt2Id` – Converts an extended field ID to an ordinary ID.
- `fieldId2Ext` – Converts an ordinary field ID to an extended ID.
- `fieldExt2Idx` – Returns the array index of an extended field ID.

Non-array fields also have an array index and are treated as an array field that contains only one element, meaning that their array index is 1. This index can hold the value 0 or 1. The same field can have two different field IDs, one being 1<<16 bigger than the other. These two IDs can be used interchangeably, and the system processes them as if they were identical.
It is best practice to use the `fieldExt2Id` method to remove the array index part of a field ID, if it is not needed.

See Also
Best Practices: Table Fields

Best Practices for Indexes

The basic rules for index design are as follows:

- Assign a unique index to each table.
- Add as few indexes as possible and maintain query performance.
- Strongly consider designating one of the indexes as the cluster index.

An error will be displayed if an index is overlapped by another index, and the other is enabled and doesn't have a configuration key. An error will also be displayed if an index is created with no fields in it.

Using the Key
If the table has a key, create a unique index on the fields in the key (set the `AllowDuplicates` property to No). The database system ensures the uniqueness of the key.

When to Create an Index
The advantages of indexes are as follows:

- Their use in queries usually results in much better performance.

- They make it possible to quickly retrieve (fetch) data.
- They can be used for sorting. A post-fetch-sort operation can be eliminated.
- Unique indexes guarantee uniquely identifiable records in the database.

The disadvantages of indexes are as follows:

- They decrease performance on inserts, updates, and deletes.
- They take up space (this increases with the number of fields used and the length of the fields).
- Some databases will monocase values in fields that are indexed.

You should only create indexes when they are actually needed.
Take care not to add an index on something that has already been indexed. If you need a more detailed index, you can add fields to an existing index as long as it is not a unique index.

Tip:
It is more time-consuming to change fields at the beginning of an index than at the end of an index. If fields in an index are updated frequently, place these at the end of the index.

Index Hints
When you use index hints, verify them with performance tests. The optimizer might be able to find a more efficient hint.
The following examples show finding ledger transactions in account number, transaction date order.
Last weeks' (few days) transactions on all the (many) Profit & Loss accounts.

```
select ledgerTrans
    index hint DateIdx
    order by accountNum, transDate
    where ledgerTrans.accountNum >= '40000'
        &&  ledgerTrans.accountNum <= '99999'
        &&  ledgerTrans.transDate  >= 26\04\1999
        &&  ledgerTrans.transDate  <= 02\05\1999;
```

Transactions for the whole year (many dates) on (the few) liquid assets accounts.

```
select ledgerTrans
    index hint ACDate
    order by accountNum, transDate
    where ledgerTrans.accountNum >= '11100'
        &&  ledgerTrans.accountNum <= '11190'
        &&  ledgerTrans.transDate  >= 01\07\1999
        &&  ledgerTrans.transDate  <= 30\06\2000;
```

See Also
Best Practices for Index Properties
Clustered Indexes
Unique Indexes

Unique Indexes

This topic describes the best practices related to unique indexes.

Unique Indexes and RecID
If a table is not given a unique index (because there is no key in the application), the system will create a unique index to get a key. The index will consist of the fields in the shortest index definition, measured in bytes, appended with the RecId.
You can make any index unique by appending the RecId to the index.

Create an index on RecId only if you need it. For example, if the RecId field is used for lookup operations. A best practice warning appears if you create an index on RecId without using RecId for any explicit lookup operation. A best practice warning appears if RecId is used for a lookup when no index has been created on RecId.
If needed, create a unique index by setting the `CreateRecIdIndex` property to Yes. The default is to set it to No, to save disk space and insert/update time.

Adding or Changing Unique Indexes
If you add a new unique index to a table, or change an existing one, it will cause problems for users when they upgrade to a new version of Microsoft Dynamics AX. This will cause a best practice error.
This error can be fixed by implementing an upgrade script called `AllowDup`*TablenameIndexname*, or `DeleteDup`*TablenameIndexname* as a pre-synchronization upgrade job. For example, if the new unique index on MyTable is called NewUniqueIndex, the script should be called `AllowDupMyTableNewUniqueIndex` or `DeleteDupMyTableNewUniqueIndex`.

"AllowDup" scripts
Use this to temporarily disable the unique index. When you have removed conflicting fields, you need to run an upgrade script to re-enable the unique index.
"AllowDup" scripts should contain the following code.

```
{
DictIndex  dictIndex = new DictIndex(
    tablenum(TableName),
    indexnum(TableName, IndexName));
    ;

    ReleaseUpdateDB::indexAllowDup(dictIndex);
}
```

"DeleteDup" scripts
Use this to delete all conflicting fields.
"DeleteDup" scripts should contain the following code.

```
{
    ;
    ReleaseUpdateDB::deleteDuplicatesUsingIds(
        tablenum(TableName),
        fieldnum(TableName, UniqueIndexField));
}
```

See Also
Best Practices for Indexes

Clustered Indexes

Organizing your tables with a clustered index usually has performance advantages. The general rules are as follows:

- If only one index is created on the table, make it clustered.
- Create a clustering index on the key on all group and main tables.

Note:
The `ClusterIndex` table property determines which index on a table is clustered.
Carefully consider how to cluster your transaction tables. They have many records and receive many database operations. There is a great potential to improve speed and reduce memory usage.

It is advantageous if records are usually inserted at the end of the index, for example, if the index contains the current date as part of the key.

The advantages of having a cluster index are as follows:
- Search results are quicker when records are retrieved by the cluster index, especially if records are retrieved sequentially along the index.
- Other indexes that use fields that are a part of the cluster index might use less data space.
- Fewer files in the database; data is clustered in the same file as the clustering index. This reduces the space used on the disk and in the cache.

The disadvantages of having a cluster index are as follows:
- It takes longer to update records (but only when the fields in the clustering index are changed).
- More data space might be used for other indexes that use fields that are not part of the cluster index if the clustering index is wider than approximately 20 characters).

Avoid clustering index constructions where there is a risk that many concurrent inserts will happen on almost the same clustering index value. They will be directed to the same page, resulting in more frequent page splits, which will result in locks and thus lead to poorer performance. Ensure that the inserts are distributed throughout the clustering index.

Note:
Clustered indexes are referred to as Index Organized Tables in Oracle and Cluster Indexes in SQL.

See Also
Best Practices for Indexes

Best Practices for Index Properties

The following table describes the standards and best practices in Microsoft Dynamics AX for setting the properties for an index. For a description of each property, see Index Properties.

Property	Rules
`ID`	Always ship a table index with the same ID as it has been shipped with before. If you attempt to create a new index with an ID that has already been used for an index in Microsoft Business Solutions—Axapta, an error occurs.
`Name`	Use the field name for a single field index. You must use "Idx" as a postfix for index names. If you attempt to create an index with a name that has already been used for an index in Microsoft Business Solutions—Axapta, an error occurs.
`AllowDuplicates`	The primary index must have the `AllowDuplicates` property set to No. There should be one unique index per

Property	Rules
	table for tables that belong to the table groups Group, Main, and WorksheetHeader (set `AllowDuplicates` to No).
`Enabled`	This property is usually set to Yes.
`ConfigurationKey`	You should use a configuration key so that the index is disabled for features that are disabled.

Note:
If forms are opened with no specific ranges or specific sort orders, and no index is specified in the index property of the form's data source, forms sort on the first index.

See Also
Best Practices for Indexes

Best Practices for Table Relations

You will get a best practices error if you create a relation on a table but do not add any fields to it.

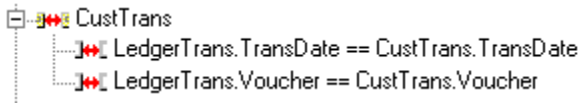
Name
The name of a relation should be postfixed with the name of the table it relates to. For example, CustBankAccounts could be a relation on the BankAccounts table.
If there is only one relation for a table, you can just use the table name for the relation instead.

Relations in the Data Model
Relations in the data model must be expressed in relations that are defined on the extended data types of the fields in the tables or explicitly on the tables. Such relations must have the `Validate` property set to Yes. Multi-field relations must be specified on the table.
A relation should be defined on the table that is holding the foreign key to the relating table.
The system guarantees that data entered in the database fulfills the specified relations.

The Self Relation
If a table has a key, the key must be defined by using relations. Such a relation is called the 'self relation'.
The self relation should not be set up in situations where the fields in the key are all foreign keys (such as relations to other tables) - or more specifically, where the last field in the self relation is a foreign key.

Navigational Relations
Navigational relations are definitions of relations that exist among data in tables where there are no integrity constraints.
Defining a navigational relation benefits the system's Auto Join system when one form is opened from within another form.
Define navigational relations to make it easy to navigate in the application by using related information.
A navigational relation has the `Validate` property set to No.
The Auto Join system uses relations that are defined at both the involved tables, which means that you only have to define a navigational relation on the one table to the other to make the Auto Join system work both ways.
Example

The CustTrans relation on the LedgerTrans table is an example of a purely navigational relation.



Both tables hold information about TransDate and Voucher, but neither of them are keys that are used for validating the other.
You can open a CustTrans table-based form from a LedgerTrans table-based form. The system will only show the CustTrans records that have the same TransDate and voucher as the LedgerTrans record has in the form from where you started.
The relation also makes it possible to start from the CustTrans table based form located on a record with a particular transDate and voucher and to open a LedgerTrans table based form that will show only the records with the same TransDate and voucher.

Relationships Involving Record ID Fields
Do not create a table relationship for a single field, if the field is derived from `RefRecId`.
Create the relationship on the extended data type for that field instead.

Configuration Keys
The configuration key that is specified for the field in a relation must be the same key, or in the same tree of keys, as the key used for the related field. If the external field in a relationship can be removed, due to configuration key settings, and the field on which the relation is set is not also removed, you will get a best practices warning.

Best Practices for Table Methods

Following are the types of Table Methods:
- Application-defined methods – created by a developer specifically for a particular table.
- System-defined methods – automatically available on every table. These methods are inherited from the `xRecord` system class.

Application-Defined Methods
Use the properties available on tables and table fields rather than creating code for the same purpose. If you do need to create a method on a table, it must be directly related to that table.
If a table has a unique key, create the following methods:
- `staticfind`
- `staticexist`
- `static checkExist`
- `static txtNotExist`

By default, these methods run on both the client and the server. You can state this explicitly in the method declaration (`client server`), but don't specify a single tier (`client` or `server`). It is a best practice to create these methods on tables. A best practices error does not occur if the method is not called.

System-Defined Methods

System-defined method	Rules
`clear`	Called by the kernel—you do not need to call this method from the application.
`delete`	Secure any related transactions. If you have DeleteActions on one table set to Cascade delete on another table,

System-defined method	Rules
	avoid writing code on the `delete` method on the second table (for performance reasons). If you don't add code here, the database management system can quickly do cascading deletes by using direct SQL deletes (`DELETE_FROM`). If you do add code here, the system creates a `while select` statement, and then executes the `delete` method on all child tables.  If you are deleting from the buffer, use `skipDelete`.
`doInsert`, `doUpdate`, `doDelete`	Avoid using these methods. These methods should only be used under strict control because they bypass the following: • Any validations that have been set up. • Any code that was written in the `insert`, `update`, and `delete` methods.
`helpField`	When manipulating an array, use the standard methods on the `Global` application class as follows: • `fieldExt2Id` • `fieldId2Ext` • `fieldExt2Id` For more information about the field ID of an array, see Extended Field IDs.
`initValue`	The call to `super()` assigns the default values that you have set by using the record template. Use the method to assign initial/default values to a new record.
`insert`	Secure any related transactions with `tts`. If `insert` is overridden, Optimizing Record Inserts reverts to record-by-record insert.
`isFormDataSource`	Called by the kernel—you do not need to call this method from the application.
`merge`	Called by the kernel—you do not need to call this method from the application.
`modifiedField`	Called by the kernel—you do not need to call this method from the application.
`postLoad`	Avoid placing any code here. Instead, use display methods. The `postLoad` method is called every time a record is fetched from the database. Avoid creating any code on this method that will have an impact on performance. For example, avoid adding code that involves calls between the client and

System-defined method	Rules
	server, or any code that results in a database operation.
`reRead`	Called by the kernel—you don't need to call this method from the application.
`toolTipField`	When manipulating an array, use the standard methods on the `Global` application class as follows: • `fieldExt2Id` • `fieldId2Ext` • `fieldExt2Id` For more information about the field ID of an array, see Extended Field IDs.
`toolTipRecord`	Enables you to append additional information to the standard tooltip text. For example, use the `toolTipRecord` method to include a message to display a "customer out of credit" message.
`update`	Secure any related transactions with `tts`. If `update` is overridden, Optimizing Record Inserts cannot be used, and automatically reverts to record-by-record insert.
`validateDelete`	Do not delete the `super()` call if you override this method. The method should return a Boolean—don't throw an exception here.
`validateField`	You should respect the task performed by the `super()` call. The method should return a Boolean—don't throw an exception here. When manipulating an array, use the standard methods on the `Global` application class as follows: • `fieldExt2Id` • `fieldId2Ext` • `fieldExt2Id` For more information about the field ID of an array, see Extended Field IDs.
`validateWrite`	Respect the task performed by the `super()` call. The method should return a Boolean. Don't throw an exception here.
`xml`	Called by the kernel—you don't need to call this method from the application.

See Also
Best Practices for Methods

Views Best Practice Checks

This topic contains information about the best practices for setting view properties and view field properties.

View Properties

The properties that are available for views are a subset of the properties available for Best Practices for Table Properties. Views are read-only, and so some of the properties are set by default and cannot be changed. Best practices are listed for some of the properties that can be edited:

Property	Rules
`Name`	Cannot have same name as a class, table, map, enum, or data type. Prefix with the module name, for example: Cust, Ledger, Proj. Infix with a logical description of contents, for example ProjCommittedCostCategoryView, where "ComittedCostCategory" is the infix. You can use 'View' as a postfix.
`Label`	Mandatory property. The text value of the Label must be unique across all tables, views, and maps, in all languages.
`TitleField1, TitleField2`	`TitleField1` is the title that is used on forms that use the view as main datasource. `TitleField2` is the secondary title and is used on forms that use the view as main datasource.
`AnalysisVisibility, AnalysisSelection, TypicalRowCount, IsLookup, SingularLabel`	These properties are inherited from tables, where they are used as part of the end-user reporting system. End-user reporting is not implemented for views, and so any best practice errors about these properties should be ignored for views.

View Field Properties

Many of the properties that are available for view fields are also available as properties on table fields. The best practice advice is the same as for table field properties.

See Also

Best Practices for Table Field Properties

Extended Data Types Best Practice Checks

The basic rules for using extended data types are as follows:

- Use an extended data type wherever possible.
- Create an extended data type for each atomic entity in the situation that your application is modeling.
- The hierarchies you create among the extended data types should be able to pass the "is-a" test (except for the super-grouping types and the mixed field types described later in this topic). That is, an extended data type should be a "member" of the parent extended data type concept. For example, SuppItemId "is-a" ItemId, but SuppItemId is not a ClassName.

- Only create subtypes when they are needed. If you want to add a field to a table or to a dialog, do not choose a type that is too general, and just add texts for label and help locally. Instead, create the type from the beginning and reuse it everywhere else.
- Use extended data types for setting up relations between two tables by using RecId.
- Do not directly use the system data types `recID` or `tableID`. Use the extended data type `RefRecId` for types that refer to record IDs, and `RefTableID` for types that refer to table IDs, or types derived from these.

For more information, see Best Practices for Extended Data Type Properties

Super-Grouping Types used as Templates for other Types
Extended data types like `SysGroup` and `Description` are used for defining a uniform user interface in the standard application. All group table identifiers and names are initially set up with the same width. Specialized extensions should be created for each usage.
These super-grouping types can be used for setting the length and so on for all the fields that use their subtypes, but the intention is that they must not be counted on as compatible.
The specialized subtypes can be disintegrated from their super-grouping types at the site of the implementation. This allows for customized settings of each actual type.
The super-grouping types should not be used from X++ code and in fields because their subtypes can be disintegrated. They must only be used as super types.

Mix-Typed Fields
Sometimes it is necessary to use one field to hold foreign keys to different tables that have keys defined with completely different extended data types. In such cases, you must make sure that the field of the foreign key can hold all possible values of all possible keys that are potentially going to be stored in it. When you have a key for a string field, ensure that the foreign key is at least as wide as the widest key that can potentially be held in it.
Such foreign key fields will always be supported by an `Enum` type field that is used to decide what kind of key is going to be stored in the field and used in the relation setup as "This fixed" to support the validation of the field contents and so on.
Following are examples:
- \Data Dictionary\Tables\PriceDiscAdmTrans\Fields\ItemRelation (supported by ItemCode)
- \Data Dictionary\Tables\PriceDiscAdmTrans\Fields\AccountRelation (supported by AccountCode)
- \Data Dictionary\Tables\LedgerJournalTrans\Fields\AccountNum (supported by AccountType)

Configuration Keys and Relationships
The configuration key specified for the extended data type in a relation must be the same key, or in the same tree of keys, as the key used for the related field. If the external field in a relationship can be removed, due to configuration key settings, and the extended data type on which the relation is set is not also removed, a best practices warning occurs.

See Also
Best Practices for Extended Data Type Properties

Best Practices for Extended Data Type Properties

In Microsoft Dynamics AX, the Best Practices for extended data type properties are listed in the following table. For more information about properties, see Extended Data Type Properties.

Property	Rules
ID	Always ship an extended data type with the same ID as it has been shipped with before. If you attempt to create a new extended data type with an ID that has already been used by another extended data type, an error occurs.
Name	The names of extended data types should reflect the real-world items they model. If the extended data type belongs to an application module (such as customers or vendors), the name must be prefixed with the name of the module (Cust, Vend, and so on). If an extended data type extends another extended data type, its name can be the name of the parent followed by its own specialization. If it is used in another module, however, it can be prefixed. If it is a data type for identification purposes, such as for a key field in a table, the name must be postfixed with "Id." For example, `JournalId`, `InventJournalId`, `CustGroupId`, and `VendId`. Other common postfix terms can also be used, but avoid "Code," "Num," and "Type." If you attempt to create an extended data type with a name that has already been used, an error occurs.
Label	Do not set the `Label` property to the same value as the `HelpText` property. The label should be defined at the most generic place, and not be duplicated down the hierarchy. Use the `LabelId` extended data type.
HelpText	Do not set the `HelpText` property to the same as the `Label` property. Make it more descriptive and helpful. The `HelpText` property should be defined at the most generic place, and not be duplicated down the hierarchy. For instance, if an extended data type inherits from an enumeration, the `HelpText` property should be reused, not duplicated. Use the `LabelId` extended data type.
FormHelp	Set `FormHelp` only when the standard lookup system facilities are not useful.
AnalysisDefaultSort	Set to Descending if the values in this field are more commonly sorted in

Property	Rules
	descending order, for example, the date on which an e-mail was received.
AnalysisGrouping	Set to Discouraged if the values in the fields with this type are likely to be unique, for example, phone numbers.
DisplayLength	Set to Auto.
Extends	If a part of a hierarchy, it must inherit an "is-a" type.
DisplayHeight	Set to Auto.
SignDisplay	Usually set to Auto. Can be set to None (if used with DisplaceNegative).
DisplaceNegative	If you need negative adjustment, the standard value is 10.
EnumType	If an extended data type is of type enum, it is mandatory to set the EnumType property.
ShowZero	If the value 0 (zero) in fields of this type actually means null/nothing, ShowZero can be set to No.
ThousandSeparator, DecimalSeparator, DateFormat, DateSeparator, DateYear, DateMonth, DateDay, TimeFormat, TimeHours, TimeMinute, TimeSeconds, TimeSeparator	Set to Auto.

See Also
Extended Data Types Best Practice Checks

Base Enums Best Practice Checks

This topic describes best practices for using Enums. For rules about setting Enum properties, see Best Practices for Enum Properties.
Define and use enums when a field or a variable can have a limited, fixed number of predefined outcomes. If there might be an unlimited number of user-defined outcomes, use a to-be-related-to table instead.
If you want to make an enum a mandatory field on a table, make the first outcome with the value zero, as none, with the label **Not selected**.
Most enums can be regarded as specialized types, and should be used as they are. Some enums (like NoYes and TableGroupAll) can be regarded as more general types that can be specialized by using extended data types.

Enums and Constants

When you are working with enums and constants:

- Always let the constant be an enumerator.
- Never use numeric constants instead of enums.
- Never use other constants (such as other enum types) instead of enums (Booleans are compatible with all enums).
- Do not make Boolean tests on enum fields, except where the 0/false value is clearly the one with the false outcome.
- Never use relational operators on enums.
- Never compare or assign to enums of different types.
- Do not expect the enumerators to have a numeric value in the range of 0.<num of enumerators - 1>.

Deleting Enumerators

If you want to delete an enumerator, determine whether it has been used in persistent storage in existing tables; old data must be taken into account.
To remove the enumerator, while saving it in a database at the customers site as a table field:

1. Rename the enumerator to `DEL_<original name>`.
2. Remove its label, and replace it with the text "*** Outdated ***".
3. Add the Configuration Key `SysDeletedObjects40` ('To be deleted after update') to the enumerator.
4. Remove the usage of the enumerator from the code.
5. Write a release update job that removes the data that is based on this enumerator from the installation, probably changing it to something else, so that the total result will be consistent.

After the update, the user will not see the enumerator in the user interface.

6. Remove the enumerator in the next version so that other enumerators do not change their numerical value: to set the `UseEnumValue` property to Yes, on the enum.

See Also

Best Practices for Enum Properties

Best Practices for Enum Properties

The best practices for enum properties are shown in the following table. For more information about the properties, see Base Enum Properties.

Property	Rules
`ID`	Always ship an enum with the same ID as it has shipped with before. If you try to create a new enum with an ID that has already been used for an enum in the previous version of Microsoft Dynamics AX, you will get an error.
`Name`	An enum name must either indicate the possible enum values or indicate the type of the enum value. Examples of enums that are named according to the possible values are `InclExcl` and `NextPrevious`. Examples of enums that are named according to the type of the enum value are `ArrivalPostingType` and `ListStatus`. If you try to create an enum with a name that has already been used for an enum in the previous version of Microsoft Dynamics AX, you will get an error.
`Label`	Mandatory.
`Help`	Mandatory. The `Help` property should not be identical to the `Label` property; give the user a more thorough explanation.
`DisplayLength`	Set to Auto (to allow IntelliMorph features).

See Also
Base Enums Best Practice Checks

Configuration and Security Keys Best Practice Checks

Configuration keys determine which features are turned on during the installation. Security keys determine which features that a user group has access to (from the installed set of features).

Security Keys
Microsoft Dynamics AX consists of a number of modules. For example, General Ledger, Project, and Administration. To make it easier for an administrator to set up security keys, they are organized the same way for all modules. Only nine security keys are allowed for each module in the Navigation Pane.
Enterprise Portal security keys relate to the different Enterprise Portal roles (Administrator, Sales, Vendor, and so on). Only five security keys are allowed for each role.
Using Security Keys
The security key specified for objects in the Navigation Pane must match the location of the object. For example, items that use the BasicReports security key must be located in **Basic** > **Reports** in the Navigation Pane.
If your code needs to check access permissions, match these locations as early as possible. Provide a clear error message to inform the user at the beginning of a process if they do not have permission to execute it.
Security Key Properties

Property	Rules
ID	Always ship a security key with the same ID as it has been shipped with before. If you try to create a new security key with an ID that has already been used for a security key in Microsoft Dynamics AX, an error will occur.
Name	One of the nine security keys on a branch (the parent) should take the name of the module. For example, BOM. The other keys (up to eight more on a branch) should have the name of the module followed by one of the following suffixes: Daily, Journals, Inquiries, Reports, Periodic, Setup, Misc, or Tables. For example, BOMReports, BOMSetup, and LedgerPeriodic. Enterprise Portal keys should have a prefix of **EP** followed by the name of the role. For example, EPAdmin and EPConsultant. Additional security keys for the role should take one of these suffixes: Misc, Info, Report, or Task. For example, EPAdminInfo and EPConsultantTask. Application Integration Framework (AIF) keys should be the same as the name used for the service. The format is the module that the service is contained in,

Property	Rules
	the document name, followed by **Service**. For example, in the Sales module, SalesSalesOrderService. If you attempt to create a security key with a name that has already been used for a security key in Microsoft Dynamics AX, an error will occur.
`Label`	Mandatory property.
`AnalysisVisibility`	Mandatory property for top-level security keys (keys that have no parent key). Set to High for any key that corresponds to a core module in Microsoft Dynamics AX, for example, Ledger. Set to Low for keys associated with tables that will not usually be used for reporting. Set to None for keys associated with system functionality that should not be shown for end-user reporting.

Configuration Keys

Configuration keys should be defined so that the installation can be set up with only the features needed for each particular installation. By disabling configuration keys, administrators can reduce the potential surface of attack, thereby helping to increase the security of their Microsoft Dynamics AX installation.

Configuration keys are often arranged in hierarchies of functionality, mirroring the hierarchies of functionality in the application.

Property	Rules
`ID`	Always ship a configuration key with the same ID as it has been shipped with before. If you attempt to create a new configuration key with an ID that has already been used for a configuration key in Microsoft Dynamics AX, an error will occur.
`Name`	Follow the standard Naming Conventions. If you attempt to create a configuration key with a name that has already been used for a configuration key in Microsoft Dynamics AX, an error will occur.

Table Collections Best Practice Checks

Several standard table collections have been created in the standard application:

- Batch
- CCDataLink
- Global
- InterCompany

These table collections must be updated by any solution that changes or adds new tables to the application. By using these collections, the most common shared data or virtual company situations can be set up at the installation.
Add tables to the Global collection if the data is not company-specific. Examples are State and Unit.

There must not be a foreign key in a table in a virtual company, where the key is in the (non-virtual) company.

To get the most use of the standard table collections, it might be necessary to adjust the data model. For example, you might have to split data that would all be in one table if all data was stored under one company.

Perspectives Best Practice Checks

This topic contains best practice information for setting perspective properties, perspective field properties, table reference properties, and view reference properties.

Perspective Properties

Property	Rules
`Label`	Mandatory property. Do not set the `Label` property to the same property as the `HelpText` property.
`HelpText`	Mandatory property. Do not set the `HelpText` property to the same property as the `Label` property; the user should be provided with more detailed information than is available in the label. For more information, see HelpText Guidelines.
`ConfigurationKey`	This property must have a value, to allow the perspective to be disabled.

Perspective Field Properties

Property	Rules
`DataField`	Mandatory property. Must be selected by using the drop-down box next to the property.

Table Reference Properties

Property	Rules
`TableName`	Mandatory property. Must be selected by using the drop-down box next to the property.

View Reference Properties

Property	Rules
`ViewName`	Mandatory property. Must be selected by using the drop-down box next to the property.

Macros Best Practice Checks

Macro names should start with an uppercase letter.

Constant Values

For constant values that are used in multiple places in your code, you should use `#define` to define a macro that has the constant value. If you must change the value later, you have to change only the one macro definition. You would not have to find and edit every location where the value is used.
If the constant applies beyond the range of one class and the classes that extend from it, the macro definition must be in a macro library. Macro libraries are stored in the Application Object Tree (AOT) under the **Macros** node.

X++ Statement Substitution

The `#localmacro` directive is the best choice when you want to substitute part or all of an X++ statement into several locations in your code. For example, a recurring part of an SQL `where` clause can be reused as a macro.

Classes and Methods Best Practice Checks

This section of the SDK describes best practices for classes and methods.

- Best Practices for Classes
- Best Practices for Methods
- Best Practices for Interfaces
- Microsoft Dynamics AX Class Design Patterns

Best Practices for Classes

Use the following guidelines when you construct classes:

- The Microsoft Dynamics AX guidelines for class declarations, constructors and destructors, methods, X++ programming, and Where to Place the Code
- The Microsoft Dynamics AX Class Design Patterns
- Commonly accepted theory within object-oriented programming, including C# programming

Objects in the class should be stable, and they should be easy and predictable to use and construct.
When you design your class, it should fall into one of the following concepts:

- A class that represents an object
- An action class
- A supporting class

See Also

Best Practices for Methods

Best Practices for Class Declarations

Class Properties

Property	Rules
`ID`	Always ship a class with the same ID as it has been shipped with before. If you try to create a new class with an ID that has already been used for a class in the previous version of Microsoft Dynamics AX, you will get an error.
`Name`	Prefix: Module short name, for example *Invent*Update. Infix: Logical description of action performed or type of contents, for example Invent*Update*. Follow the general Naming Conventions. Classes that are the basis for a subsystem hierarchy should be postfixed 'Base'. For example: `RunBase`, `SysBestPracticesCheckBase`. If you try to create a class with a name that has already been used for a class in the previous version of Microsoft Dynamics AX, you will get an error.

Class Declaration Layout

```
[public] [final] class ClassName [extends SuperClassName] [implements interface1[, interface2
..]]
```

Object Member Variables

Object member variables are variables in the class declaration. Create them only if the variable cannot be created in a method. Object member variables must be the variables holding the state of the object.

Tips:

- Do not create object member variables that do not hold the state of the object. Pass these values as arguments.
- Do not create an object member only because a variable of that type and name is needed in more than one method in the object. Create the variable in each place it is needed.

Clean Up Unused Variables

Clean up unused variables in your class declaration. Right-click the class in the application object tree (AOT) and choose **Add-Ins** > **Check Best Practices**.

Class-Wide Constants

If you have to declare some constants that will be used in more than one method in the class or its subclasses, declare these constants in the class declaration (by using the #define technique).

See Also

Best Practices for Constructors
Best Practice for Destructors (finalize)

Best Practices for Constructors

Following is information about how to create a clean inheritance model and minimize problems when code is upgraded:

- Each class must have a single public construction method unless the class is abstract. If no initialization is required, use a static `construct` method. Otherwise, use a `staticnew…` method (the default constructor (`new` method) for the class should be protected).
- Each class should have at least one static construct method method.
- Each class should have at least one static new… method.
- Each class should have a new method (the default constructor). This method should be protected.
- Create accessor methods to get and set class variables.
- Create init methods to carry out any specialized initialization tasks that should be carried out after instantiation.

See Also

Best Practices for Class Declarations
Best Practice for Destructors (finalize)

Best Practices for new and static new... Methods

The `new` method is the default constructor for a class. The `new` method should be protected. Do not use it to instantiate the class. Use a `construct` or a `static new…` method instead.
It is recommended that you do not have parameters on the `new` method—use `static new…` methods instead.
X++ does not support method-name overloading. You must create your own individually named `static new…` methods with different parameter profiles. This enables you to construct a class in more than one way. Similarly, instead of creating default parameters in a `new` method, create a different `static new…` method for each possible parameter profile.
If you have created the `new` method on a subclass, call `super()` to carry out any necessary initialization that might be implemented in the superclass. The call to `super()` should be the first statement in the method.

static new... Methods

Create one or more `static new…` methods to instantiate your class.
These new methods have the following characteristics:

- Are public
- Are static
- Have a name prefixed with "new"
- Are named according to the parameter(s) they take, or logically
- Usually take only nondefault parameters
- Always return a valid object of the class's type (instantiated as well as initialized), or throw an error
- Use a construct methods to create an instance of the class
- Use accessor methods to set the class variables

The `static new…` methods have a body that contains the following structure.

```
MyClass myClass;
;

// The construct method is used to create an instance of the class.
myClass = MyClass::construct(...);

// Use accessor methods to set the class variables.
myClass.parmOneValue(...);
```

```
myClass.parmAnotherValue(...);

if (!myClass.init())
{
    throw error("Label text explaining why object was not created");
}
return myClass;
```

See Also
Best Practices for Constructors
Best Practices for init Methods
Best Practices for Accessor Methods

Best Practices for Static Construct Methods

You should create a static `construct` method for each class. The method should return an instance of the class.
The `construct` method must be:

- static
- named "construct"

In most cases the construct method should be public. If the class is instantiated using a `newParameter` method, then declare the construct method as private. 73The method should return an instance of the class and contain no other code. `construct` methods should not have any parameters, or else should have the same parameters as the default `new` constructor (it is recommended that you do not have parameters on `new`).
If your class declaration contains parameters, use a `static new` method as the constructor and use the `construct` method within the `static new` method to create an instance of the class.
For partners and customizers, this is the point to add construction functionality for new subclasses (in higher layers), without mixing code with the `construct` method in the original layer.

Tip:
You can use a code editor script to create the `construct` method for you. In the code editor, press Alt + M to open the editor scripts menu, and then select **template** > **method** > **construct**.

Example

```
static CustPaymManFileOpen construct()
{
    return new CustPaymManFileOpen();
}
```

See Also
Best Practices for new and static new... Methods
Best Practices for Constructors

Best Practices for init Methods

If you need to carry out any special initialization tasks after class instantiation and after the setting of class variables by using accessor methods, such logic should be placed in a private method called `init`.
`init` methods should be protected, and should have a void return type, or else a Boolean return type if initialization can go wrong, so that an error can be thrown.

If you rely on the user of the class to do one of the following, then you should implement `static new...` methods, so the mandatory initialization information can be supplied as specific parameters on these methods:

- Call some specific methods to initialize some member variables, after the user has new'ed the object, to completely initialize the object.
- Supply the parameters that are actually needed to your new method, having a lot of default parameters

If the `new` method has some extra initialization logic that is always executed, you should put that in the `init` method.

See Also

Best Practices for Accessor Methods
Best Practices for new and static new... Methods
Best Practices for Constructors

Best Practices for Accessor Methods

In Microsoft Dynamics AX classes, member variables are always protected; that is, they cannot be accessed directly from outside the class. They can only be accessed from within the objects of the class itself or its subclasses. To access the variables from outside, you have to write accessor methods.

Accessor methods can set, get, or get and set the value of a variable.

Accessor methods can be public or protected, and should have the same name as the member variable they access, prefixed with "parm." For example, the following accessor method gets and sets the `MyVar` variable.

```
public MyType parmMyVar(MyType _myVar = MyVar)
{
    ;
    MyVar = _myVar;
    return MyVar;
}
```

If the method needed only to get the value of the variable, it would be as follows.

```
public MyType parmMyVar()
{
    ;
    return MyVar;
}
```

When variables contain huge amounts of data (for example, large containers or memo fields), it is recommended that you use the technique in the following example. The disadvantage of using it in all cases is the overhead of an additional method call.

```
container parmCode(container _code = conNull())
{
    if (!prmIsDefault(_code))
    {
        code = _code;
    }
    return code;
}
```

Note:

You can use a code editor script to automatically create the accessor method. In the code editor, press ALT+M to open the editor scripts menu, and then select **Template** > **Method** > **parm**. Enter the data type, and then the name of the variable.

See Also

Best Practices for Constructors
Best Practices for init Methods

Best Practice for Destructors (finalize)

The X++ language has no destructor method. A destructor method is one that is called by the system when there are no more references to an object and the system starts to destroy the object.
The `Object` class has a `finalize` method, but `finalize` is empty and the Microsoft Dynamics AX system never calls it. You can override the `finalize` method if you want a method to contain code that might otherwise belong in a destructor method. However, `finalize` is not called unless your code calls it explicitly.

See Also
Best Practices for Class Declarations
Best Practices for Constructors

Best Practices for Methods

Methods should:
- Be logical.
- Do a specific task.
- Have no side effects.
- Be well structured, especially when it comes to good places for overriding and for overlayering.

Make your methods small and logical. If you have a method that does more than one 'thing'; then consider splitting it up in two (or more) methods. It will then be easier to override or overlayer exactly the functionality that needs to be specialized or customized.
Do not have any unused variables in your methods.
For more detail about best practices for methods, see:
- Best Practices for Method Modifiers
- Best Practices for Local Functions
- Best Practices for Table Methods
- Naming Conventions for Methods
- Best Practices for Parameters

Best Practices for Parameters

The names of the formal parameters in methods must be prefixed with an underscore.
If there is more than one parameter, list each parameter on a separate line and indent them with four spaces. Align the types and the parameter names horizontally.
For example:

```
DialogField addField(
    int         type,
    FieldLabel  label = '',
    FieldHelp   help = '')
```

Never assign to parameters.
Do not have unused parameters, unless you are overriding a method or implementing an interface. You will get a best practices warning for unused parameters in `private` methods, and a best practices info for unused parameters in `public` and `protected` methods (because the parameters might be used elsewhere in the class hierarchy).

See Also
Best Practices for Methods
Best Practices for Table Methods
Best Practices for Local Functions
Best Practices for Method Modifiers
Naming Conventions for Methods

Naming Conventions for Methods

Write a clear, descriptive name for your method. If one cannot be found, consider splitting it up.
Use the general name standards, and the following preferred names for methods.

Name	Description
check*	A Boolean method that checks a condition. If the condition is not fulfilled, it must put a warning on the Infolog, and return false.
exist	Returns a Boolean that is true if a record with the supplied key exists. Typical use: Table static method.
find	Returns a record indicated by the supplied key. Typical use: Table static method.
find*	Finds a record in the table (where the method is declared). The postfix is the name of the field which is used for accessing the table, or a logical name for more fields. Typical use: Table static method.
initFrom*TableName*	This buffer is initialized with values from the buffer supplied. One argument, which is a buffer of the same type as that named in the method. Typical use: Table instance method.
initParm	Used for methods that set member variables. If the method only sets some of the variables, indicate this in a prefix to the name, for example `initParmVersDate`.
is*	A Boolean method that will check some condition. If the condition is not fulfilled, it must return false. is* cannot change the method. Information must not be sent to the Infolog.
parm*MemberVariableName*	Methods used for setting and getting the value of a member variable as a part of an object initialization. The method should have the same name as the variable, prefixed with parm.
set*	Used for methods that set value(s) in the object. The name must make it clear that the method also sets the state of some other global members. set* methods should be void or Boolean, signaling the result of the set.
update*FieldName* create*FieldName*	If a method updates or creates a record, reflect that in the name, rather than calling the method `set`*FieldName*.
validate*	Same as check*.

Note:
Methods that have the names listed in the preceding table must carry out the tasks described, and must not have any additional functionality. For example, there must be no assignments in a validate, check*, or is* method.

See Also
Best Practices for Methods
Best Practices for Parameters
Best Practices for Table Methods
Best Practices for Local Functions
Best Practices for Method Modifiers

Best Practices for Local Functions

Use private class methods instead of local functions (local methods) wherever possible.
Local functions cannot be overridden, overlayered, or reused.
Local functions follow all other standards for methods—name, style, layout, and so on. However, local functions cannot be used from outside the method where they are defined.
Local function declarations should be indented to the same level as the types in the variable declarations. Like other declarations, they should be at the top of the code block and separated from other code by a semicolon (;) on a blank line. An example follows.

```
protected void setValueQty()
{
    InventTransPosting  inventTransPosting;
    InventSum           inventSum;
    InventDim           inventDim;

    void addPhysical()
    {
        if (inventTransPosting.isPosted)
            ...
    }
    ; // Semicolon at the end of the declarations
      // ... More code
}
```

See Also
Best Practices for Methods
Best Practices for Parameters
Best Practices for Table Methods
Best Practices for Method Modifiers
Naming Conventions for Methods

Best Practices for Method Modifiers

The best practices for using method modifiers are described in the following sections. For more information, see Method Modifiers.

Client or Server

These qualifiers are used for Application Object Server (AOS) tuning, where the task is to minimize the traffic between the client and the server. They are relevant only for table methods and static class methods, because other methods (class instance methods) run where their class is instantiated.

- If a method is running on the server and only makes a single call to the client, it is okay to keep it on the server. If a method makes more than one call, move it to the client and then return it.
- If a method is running on the client and only makes a single call to the server, it is okay to keep it on the client. If a method makes more than one call, move it to the server and then return it.
- If you refer to both sides in your method, you must make the decision based on the dynamics of the calls. Maybe you can restructure the method internally or split it up in more than one method.
- Do not qualify methods as client or server if they do not use anything on that tier.
- You can use both client and server to change the execution place (to Called from) of a class static method or to document that it is decided that a table method executes best as Called from.

Public, Protected or Private

You must specify an access level for your method.

Only methods that can be used safely by the user of the class or table should be declared public. Even though methods are public by default, it is best to explicitly declare them as public, to show that they are intentionally public.

These access level specifiers affect only the compilation. You can still call a private or protected method at run time by using a noncompile time-checked call technique, so be careful.

Static

It may be appropriate to make a method `static` if one of the following apply:

- It does not use the instance member variables or fields that are defined for the class.
- It is not going to be overridden.
- It runs better on a different tier than the object itself.
- It is related to the class or table, but it does not have its origin in a single object (instance).

One advantage of `static` methods is that you do not have to spend time creating an object; the method can simply be called.

Note:

Methods that are not static are referred to as "instance methods", "normal methods", "object methods", or simply "methods".

Final

Unlike other methods, `final` methods cannot be overwritten.

Abstract

Use the `abstract` qualifier when a method has to be implemented in a subclass. Even if it has no effect, it serves as documentation.

Abstract methods can only be declared as empty:

```
abstract public container pack()
{
}
```

See Also

Best Practices for Methods
Best Practices for Parameters
Best Practices for Table Methods
Best Practices for Local Functions

Microsoft Dynamics AX Class Design Patterns

The Microsoft Dynamics AX class design patterns are as follows:

- Data + Engine class pattern
- Main class pattern
- RunBase class pattern
- Strategy class pattern

See Also

Best Practices for Class Declarations
Best Practices for Constructors
Frameworks Introduction

Main Class Design Pattern

Use the `main` class design pattern whenever a class is called from a menu item.
Do not call the `main` method explicitly from the code; call it implicitly it from a menu item.
Typically, you would:

- Instantiate the class by calling the `static construct` method.
- Call the `prompt` method to create a dialog.
- Call the `run` method, if the user clicks **OK** on the dialog box.

Most classes that have a `main` method are candidates to be implemented by the RunBase framework.

Tip You can use a code editor script to create the `main` method. In the code editor, press Alt + M to open the editor scripts menu, and then select **template** > **method** > **main**.

See Also

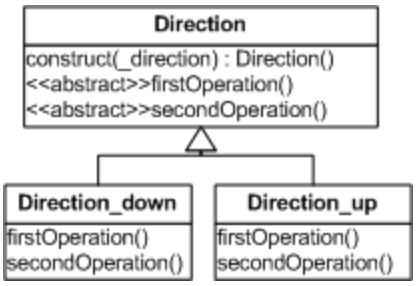
Best Practices for Static Construct Methods

Strategy Class Design Pattern

The Strategy pattern is useful if you have to do different things in a similar way; for example, if you have a lot of similar `switch` statements or `if` statements in your program.
A specific example would be similar switch statements, as follows:

```
...
// First operation
switch (_direction)
{
    case up:
        ...
    case down:
        ...
}
...
...
// Second operation
switch (_direction)
{
    case up:
        ...
    case down:
        ...
}
```

In this case, you probably have a situation where you can split up your code in a class hierarchy like this:



You should construct a correct direction object at the beginning of your code, and then work on the correct direction object in an abstract way after, as follows:

```
Direction direction = Direction::construct(_direction);
...
direction.firstOperation();
...
...
direction.secondOperation();
...
```

The correct process will occur as the direction object is instantiated to the correct situation.

See Also
Microsoft Dynamics AX Class Design Patterns

RunBase Class Design Pattern

To create a job or an Action class - a program that carries out processes, such as accepting parameters from the user and then updating records in the database - you use the RunBase framework.
The RunBase framework supports you by providing all the functions needed in such a job, and by providing the user (and the programmer) with a consistent interface.

Candidates
Most classes with a main method are candidates for being implemented with the RunBase Framework.

See Also
Microsoft Dynamics AX Class Design Patterns

Best Practices for Interfaces

The names of interfaces should end with the suffix, -able. For example: `SysMergeable` and `SysComparable`. Do not prefix interface names with "i", as in iSysComparable.

SysPackable
Classes that implement `SysPackable` must have exactly the same contents (the same variables and the same types of variables), or else you must change the version number of the container. To change the version number of the container, write the following in the class declaration:

```
#define.CurrentVersion(2)
```

(Use a number other than 1.)

SysUnitTestable
Classes that implement `SysUnitTestable` must have a name postfixed with 'UnitTest'.

Pack-Unpack Design Pattern

Use the pack-unpack pattern to save and/or store the state of an object, and then later reinstantiate the same object.

Situation

An object has a particular state. You want to work with the same object at a later time or another place (such as on another tier).

Solution

Make it possible to save the state of the object in a form that can be persistently saved or transported to another tier. Moreover, make it possible to reinstantiate the object.
For `Table` objects (records), this is straightforward because they are enabled for persistence and are automatically transported between client and server.
For class objects, create a `pack` method to read the state of the object and return it in a container suitable for saving or transporting between tiers. Reading the state of the object implies collecting the value of all its members. If the members are tables (records, cursors, temporary tables) or classes, it must also be possible to read their state.
Likewise, create an `unpack` method that takes the packed state and reinitializes an object with it. Construct the object before creating an `unpack` method.
You can also create a `static create` method that instantiates a new object. Initialize the new object with the packed state information, and return it ready for use.

Implementation

To implement the pack-unpack pattern, create the following methods.

public container pack()

Returns the state of the object as a container.
According to the persistent data storage design pattern, the first entry is a version number that describes the version of the saved structure.
Example:

```
container pack()
{
    return [#CurrentVersion,#CurrentList];
}
```

Where the macros are defined in the `classDeclaration`:

```
public class InventAdj extends RunBaseBatch
{
    InventClosing inventClosing;
    #DEFINE.CurrentVersion(1)
    #LOCALMACRO.CurrentList
        InventClosing
    #ENDMACRO
}
```

public boolean unpack(container _packedObject)

The `unpack` method takes the saved state of the object and reinitializes the object with it. It reinitializes the object members according to the values in the container, taking the supplied version number into account.
The method can return a Boolean that signals the result of the initialization process.
The object should be only instantiated before a call is made to the `unpack` method.
Example:

```
public boolean unpack(container _packedClass)
{
    int version = conPeek(_packedClass,1);
    ;
```

```
    switch (version)
    {
        case #CurrentVersion:
            [version,#CurrentList] = _packedClass;
            break;
        default:
            return false;
    }
    return true;
}
```

public void new()
The constructor (the `new` method) of the object is expected to take no parameters. It should be possible to instantiate the object without knowing anything about it except its class type, and then completely reinitialize it by using the `unpack` method.
A typical example would be to not have an explicitly defined `new` method, but to use the standard `new` method without parameters.

public static <YourClass> create(container _packedObject)
You can also create a `create` method that does the following:

- Instantiates a new object (by calling the constructor)
- Initializes it with the saved state of the object and reinitializes the object with it (by calling the `unpack` method, which can then be set to `private`)
- Returns the reinitialized object of your class

For example, this is done in the `List::create` system class method.

Limitations
An object cannot be packed unless all the members it contains can be packed.
If objects contain table or class members, it must be possible to pack these members and to return them to the same state when unpacked.
A reinstantiated object is not the same object as the one that was saved. It is simply an object of the same class whose members contain the same information.

Known Uses

- All `RunBaseBatch` descendants. The `Batch` system uses the unpacking possibility.
- `QueryRun`
- Microsoft Dynamics AX collection classes (formerly called foundation classes)

Forms Best Practice Checks

This topic describes the best practices for standard view and data-entry forms. Lookup forms are described separately. Best practices for form properties are described in the topics that are listed in the See Also section.

Form Name

The name of a form should be identical to the name of the table that is used as its data source.

Tabbed Pages

All forms should automatically open on a tabbed page.
All data entry and view forms that use tables with the table groups—group, main, or transaction—must have two tabbed pages :

- **Overview** with a grid, (@SYS9039)
- **General** with non-grid table data groups, (@SYS2952)

These tabbed pages enable the user to choose between the spreadsheet style of the grid, and the form style of the non-grid view. If you do not use the previous labels for the first two tabs of a data entry form, a best practices warning appears.
A form should open on the first tabbed page. The `Tab` property for the **Tab** control in your form design must be set to **Auto**.
Do not set the `HelpText` property for a TabPage form or a group on a tabbed page unless the form is a wizard, or unless the group has the `FrameOptionButton` property set.

Buttons

Place buttons in a **ButtonGroup** control with the following property settings:

- `Left` - Set to Right edge
- `ArrangeMethod` - Set to Vertical

Buttons and their associated menu items must fully support multi-selection in the grid or grids.
Command buttons should not be defined.

Keep Default Settings

When you design forms, their properties should retain their **Auto** or **Default** settings. This is the primary form-design rule.
Many aspects of form design are defined in the data source table. They shouldn't (usually) be overridden in the form. These form-design aspects include the following:

- Relations
- Delete actions
- `TitleField1` and `TitleField2` properties that are used by the `caption` method
- Labels used as captions on field groups, on controls, or on the table field group controls
- Order of the fields in field groups is reflected when the field groups are displayed on the forms
- AutoReport field group that specifies the initial print layout

The first index is used for sorting in the form, unless this behavior is overridden.

Enable IntelliMorph Features

You should enable all IntelliMorph features in forms. Following are examples of how to enable these features:

- Construct the form by using a menu item and a `MenuFunction` object. This is automatically done when you use menu items.
- Prioritize your data design elements (controls) as follows:
- Table field groups with the `AutoDataGroup` property set to **Yes**
  –or–
  Table field groups (with the `AutoDataGroup` property set to **No**)
  –or–
  Table fields
  –or–
  `Display` and/or `edit` methods—preferably from the data source table
  –or–
  Controls that are based on extended data types
- Ensure that labels, and to a certain extent HelpTexts, are not overridden on the controls—especially not with the same value as the one supplied from the Field, the `display` and/or `edit` methods or the extended data type.

Ensure that the Caption is not overridden on the Table field groups bound group controls, and especially not with the same value that is supplied from the Table field group.

Form Size

Forms must have the ability to be resized—the `Width` and `Height` properties on the Tab control and the **Grid** control must be set to column width/column height.
A form must fit a screen that has a 1024 x 768 resolution. Because the status bar, caption, and toolbars take up about 100 pixels vertically, and the menu bar in the Navigation Pane takes up 200 pixels, the maximum size of a form should be 824 x 668. If you exceed this size, a best practices warning appears.  If you exceed 1080 x 924, a best practices error appears.

Form Initialization

If a form cannot be started (initialized) correctly, you must inform the user and end the process. For example, if the form has to be started by using certain parameters and is not, throw an error as soon as possible. Provide a straightforward, descriptive error message that states the problem. The following example shows an error being thrown.

```
void init()
{
    if (!element.args()
        || element.args().dataset()
        != tablenum(BOMCalcTable))
        throw error(Error::MissingParameter(element));
    super();
    ...
}
```

Printing from Forms

By default, printing is available from all forms with a data source through the Auto Report facility.
If a special report has been created to support general printing from the form, implement it on the form's user interface. Override the standard `print` method on the form. The range (amount) to print must by default correspond to what is on the screen. The report prints when the Print icon is activated—an explicit **Print** button should not be added to the form. If (in rare cases) a **Print** button is needed, implement it by using a **Print CommandButton** control.
Implement the call to the report by using its menu item (depending on the functionality needed) as shown in the following example.

```
void print()
{
    new MenuFunction(menuItemOutputStr(...),
                     MenuItemType::Output).run([...]);
}
```

Reports with more specialized functionality should be added to the form by using menu item buttons.

Dimension Fields

If the table has a dimension field (a field based on the `Dimension` extended data type) that is going to appear on the form, do the following.

1. Create a separate tabbed page named "dimensionTab," positioned as the last tabbed page.
2. Place the Dimension field group as an auto data group on the tabbed page. The `FrameType` property on the group must be set to None (to avoid duplicated text: A Dimension tab with a Dimension group).

The label for all the dimension names is: @SYS14926, en-us: "Dimension."

ActiveX Control Security

ActiveX controls can be a security threat. Note the following security issues:

- ActiveX controls run in the same security context as the Microsoft Dynamics AX client.
- ActiveX controls that are used by Microsoft Dynamics AX should not be marked as safe for scripting.

If an ActiveX control in Microsoft Dynamics AX has a method with a signature and a name that are equal to a public method that is exposed by the ActiveX control, the X++ method is always called when it is referenced from X++ code.

For more information about ActiveX controls and security, see http://msdn.microsoft.com/workshop/components/activex/sec_activex.asp.

Image and Animation Files

To help prevent attackers from modifying the contents of image and animation files that are used by your form, ensure that access control lists are applied to the image and animation files.

See Also

Best Practices for Form Data Source Properties
Best Practices for Form Design Properties
Best Practices for Form Control Properties
Best Practices for Lookup Forms
No Code in Forms

No Code in Forms

Do not place code in a form unless you cannot place it in a class or in a table method. Code written on forms cannot be reused and is difficult to customize.

A class that processes all the logic in a form should have the same name as the form with "Form" as the suffix. If you have a complex form, create a class for server-related tasks. This class should have the same name as the form, but with "Server" as the suffix.

If you have code in the form, the code should be for making only simple changes to the appearance of the form or to the form's query. Try to place the code on the form or on the data source. Place code directly on controls only when you are absolutely certain that there is no other solution, and then use the `AutoDeclaration` property on these controls.

Reasons for Removing Code from Forms

- Forms are entirely client-based. There should be no code in them that manipulates the database (business logic). The code should be removed from the form and placed on the server.
- Putting the business logic in methods on classes and tables makes it possible to customize, method by method. As a result, only the customized methods are overlayered. Methods on classes and tables are stored individually in the Application Object Tree (AOT), and thus can be individually overlayered. Because a form is a single layered application object, changes to the business logic in a form overlay the complete form—not just any customized method. This results in unwanted, redundant application object content in the upper layer.
- Code placed in classes and tables can be reused in other application objects. Code inside a form is not practical to use—it is difficult to access, and there is no compile-time control.
- The physical implementation of Web forms is very different from ordinary forms. If your business logic is not dependent on the look and implementation of a form, it is much easier to duplicate the form as a Web form.
- When you want to enable your business logic for COM, it is essential that the business logic is not constructed in a form-dependent way.

See Also
Forms Best Practice Checks

Best Practices for Form Data Source Properties

Form data sources should be set up to use AutoJoin System of Microsoft Dynamics AX. The first record in the data source should be shown when the user opens the form.
Best practices for some of the properties are shown in the following table. For a description of all form data source properties, see Form Data Source Properties. For lookup forms, refer to Best Practices for Lookup Forms. You can also set properties on datasource fields.

Property	Rules
`Name`	Give the data source the same name as the underlying table. If there is more than one data source with the same table, give the data sources individual logical names that describe their purposes.
`Table`	Should be identical to `Name`.
`Index`	If you specify an index here, it will be used as an index hint on each of the queries to the database. It will specify an access path and a sort order for the records shown in the form, based on this data source. The initial sort order for the records shown in a form is prioritized by the system: • If sort fields are added to the data source query, the sort specification is used. • If an index is specified in the index property on the data source, the sort order that is implicitly specified in that index is used. • If the data source is auto joined with another data source, the system finds the most adequate index for this join and sorts the data according to that index. • If nothing else is specified, the sort order is the one that is implicitly specified in the first index (the one with the lowest ID) on the table that is used in the form data source. When no index hints are specified, it is up to the database management system to find an applicable access path that is based on the information in the supplied query. The sort order can be changed by the user, by using the different sort options that are presented on the controls and in the query dialog.

Property	Rules
`AllowCheck`, `AllowEdit`, `AllowCreate`, `AllowDelete`	Usually should be set to Yes, but you should decide what is appropriate for your application. Set to No for lookup forms. **Note:** `AllowCheck` on joined data sources (inner, outer, exist, not exist) must be set to Yes (which is the default) to enforce security checking across all data sources.
`StartPosition`	Unless your form is used for postings, the `StartPosition` property should be set to First.
`AutoSearch`	Must be set to Yes.
`AutoNotify`	Must be set to Yes.
`AutoQuery`	Must be set to Yes.
`OnlyFetchActive`	This property is designed to be used in lookup forms, but it can be used safely in any view-only form with no code or buttons to activate other application objects. Only the fields that are controls on the form will be selected from the database, meaning that the form will open more quickly
`JoinSource`	Set this property only when two or more tables are used as the data source and you want to join them.
`LinkType`	Must be set to Delayed for the outer (externally linked) data source.

See Also
Best Practices for Form Control Properties
Best Practices for Form Design Properties
Forms Best Practice Checks

Best Practices for Form Design Properties

The primary rule for form design is that all properties should keep their Auto or Default setting. Many form design properties also exist on the individual controls, for example the `Width` and `Height` properties.
Also refer to the description of design properties in:

- Form Design Properties (a general description of the properties)
- Lookup Forms (best practice for lookup forms)

Property	Rules
`Left, Top, Width, Height`	Leave these properties set to Auto.
`Caption`	Mandatory property, unless the `Visible` property is set to No, or the form is a lookup form. Set this property to the same value as the label for the table underlying the

Property	Rules
	primary data source on the form (the one set in `TitleDatasource` property). The label on the menu item that is used for activating the form, the caption of the form, and the table label must be in balance.
`TitleDatasource`	Set this property to the primary data source in the form. The form title consists of the value of the `Caption` property, and, if a `TitleDatasource` is specified: a dash and the result of the caption method on the table underlying the specified data source Examples: • Customers - Customer Account: 4010, The Lamp Shop • Customers - Customer Account: 5000, New Record When a user creates a new record, the title will be appended with the text 'New Record'.
`Frame`	Typically, the Frame type is Standard. The other types do not have a caption and are typically used when a new form is opened from within a form (for example, when a form is opened when the user clicks a lookup button ). Set to Border for lookup forms.
`WindowResize`	Keep the default setting.
`WindowType`	Set to Standard, except for lookup forms, where it should be set to Popup.
`HideToolbar`	Set this property to No for all data entry forms. Set this property to Yes for lookup forms.
`Columns`	Keep the default setting (1) for all data entry forms.
`SetCompany`	If the `SaveDataPerCompany` property on a table is set to Yes, then the `SetCompany` property on a form design that uses the table as a data source must also be set to Yes.

See Also
Forms Best Practice Checks
Best Practices for Form Control Properties

Best Practices for Form Control Properties

The basic rule for Form Control Properties is to use the system's `Auto` property values wherever you can. This ensures a uniform application interface and reduces repetitive

work. For example, the Auto settings for `Dimension` properties such as `Top`, `Left`, `Width`, and `Height` make allowances for preferences in font, font style, font size, and so on, and allow the dimensions to change dynamically when a different font is selected.
Specific rules for a few control properties are described in the following table.

Property	Rules
`Name`	All control names should be unique on a form.
`AutoDeclaration`	Use the AutoDeclaration feature on controls that you address for programming from X++ code in the form.
`AllowEdit`	If `AllowEdit` is set to No, set it as close to the table field as possible. The `AllowEdit` property is on: • Each field in a table • Each field in a form data source • The form data source (the table) • Each control on a form, on container controls like tabs, and on individual controls
`AutoDataGroup`	Set the `AutoDataGroup` to Yes on Data group on forms, if possible. This is recommended for performance reasons.
`HelpText`	Mandatory property.
`Label`	Mandatory property, for controls where this property appears.

Deactivating Fields/Controls
Fields that the user should not be able to change by using form controls should be set as shown in the following table .

Property	Value	Description
`AllowEdit`	No	Set to No to stop the user from changing the value.
`Skip`	Yes	Set to Yes if you don't want the user to enter the field while they tab through the form.
`Enabled`	Yes	Set to Yes by default. This allows the user to navigate to the field to see the Help text or to copy the field value.

See Also
Best Practices for Form Data Source Properties
Best Practices for Form Design Properties
Forms Best Practice Checks

Best Practices for Lookup Forms

Create lookup forms only in situations where the system does not create them automatically. Alternatively, you can design lookup forms by using the AutoLookUp table field group. The form name must be postfixed with 'Lookup'.
If you only need to specify the fields or the query or both, use the functionality in the `SysTableLookup` class, if possible.
Manually created lookup forms must have the same functionality and general appearance as the lookup forms that are automatically generated by the system.

- They must support a query expression in the looked up control.
- They must support "Auto find"/Focus on the rows that are found as characters are keyed in.

Manually created lookup forms can be activated from the `FormHelp` property on extended data types or from the `lookup` method on form controls.

Run Method

The code in the `run` method performs the actual filtering. To avoid flicker in the form, the default query execution is disabled in case of filtering.
In the example below, `Common_ds` is the datasource in the lookup form and `Common_LookupField` is the control in the lookup form from which the lookup value is selected (the one you used for the 'this.selectmode' call).

```
void run()
{
    FormStringControl callerControl =
        SysTableLookup::getCallerStringControl(element.args());
    boolean filterLookup = false;
    ;

    if (callerControl.text() && callerControl.hasChanged())
    {
        filterLookup = true;
        Common_ds.autoSearch(false);
    }
    super();
    if (filterLookup)
    {
        Common_ds.research();
        Common_LookupField.filter(callerControl.text());
    }
}
```

Properties

Property	Property set on:	Value	Description
`AllowCheck`	Data source	No	Security check must be switched off.
`AllowEdit`	Data source	No	Not allowed in lookup forms.
`AllowCreate`	Data source	No	Not allowed in lookup forms.
`AllowDelete`	Data source	No	Not allowed in lookup forms.
`OnlyFetchActive`	Data source	Yes	The form will select only the fields that are

Property	Property set on:	Value	Description
			on the controls in the form. Since there are no codes in lookup forms, they will open faster because the result set is smaller.
`Frame`	Design	Border	No caption will appear above the form.
`WindowType`	Design	Popup	-
`ShowRowLabel`	Grid	No	-

See Also
Forms Best Practice Checks

Best Practices: List Pages

Microsoft Dynamics AX conducts a best practices check of new or modified list pages. The best practice checks seek to establish a common look and behavior for all list pages. For more information about best practice checks, see Best Practices for Microsoft Dynamics AX Development.

Best Practice Checks
The following table lists the best practices error messages for list pages.

Message	Message type	How to fix the error or warning
List Pages must have a name that ends with "ListPage".	Warning	Append **ListPage** to the value of the form **Name** property.
List Pages must have their TopMargin property set to "Auto".	Warning	Set the value of the **TopMargin** property in the form **Design** node to **Auto**.
List Pages must have their BottomMargin property set to "Auto".	Warning	Set the value of the **BottomMargin** property in the form **Design** node to **Auto**.
List Pages must have their LeftMargin property set to "Auto".	Warning	Set the value of the **LeftMargin** property in the form **Design** node to **Auto**.
List Pages must have their RightMargin property set to "Auto".	Warning	Set the value of the **RightMargin** property in the form **Design** node to **Auto**.
List Page datasources must have their AllowEdit property set to "No".	Warning	Set the value of the **AllowEdit** property in the list page data source to **No**.
List Page datasources must have their AllowCreate property set to "No".	Warning	Set the value of the **AllowCreate** property in the list page data source to **No**.
List Page datasources must have their StartPosition property set to "First".	Warning	Set the value of the **StartPosition** property in the list page data source to **First**.
A List Page must have a single Action Pane.	Warning	If the list page does not include an **ActionPane**, add an **ActionPane** control to the **Design** node of that list page.

Message	Message type	How to fix the error or warning
		If the list page includes more than one **ActionPane**, remove all except one **ActionPane** control from the **Design** node of that list page.
An Action Pane should not be present on a form that isn't a List Page or Content Page.	Warning	Remove the **ActionPane** control from the **Design** node of the form. Or, set the **WindowType** property in the **Design** node of the form to **ListPage** or **ContentPage**.
A List Page must have a grid.	Warning	Add a **Grid** control to the **Design** node of the list page.
List Page grids must have their AllowEdit property set to "No".	Warning	Set the value of the **AllowEdit** property of the **Grid** control to **No**.
List Page grids must have their Width property set to "Column width".	Warning	Set the value of the **Width** property of the **Grid** control to **Column width**.
List Page grids must have their Height property set to "Column height".	Warning	Set the value of the **Height** property of the **Grid** control to **Column height**.
List Page grids must have their ShowRowLabels property set to "Yes".	Warning	Set the value of the **ShowRowLabels** property of the **Grid** control to **Yes**.
List Page grids must have their Datasource property set to a valid datasource.	Warning	Set the value of the **DataSource** property of the **Grid** control to the name of a data source for that list page.
List Page grids must have their DefaultAction property set to a button on the form. The DefaultAction property should normally point to a button that performs the "Open" action.	Warning	Set the value of the **DefaultAction** property of the **Grid** control. Specify the name of an action pane button.
List Page Action Panes must have their Width property set to "Column width".	Warning	Set the value of the **Width** property of the **ActionPane** control to **Column width**.
A document handling button on an Action Pane should use the label @SYS114630 for its Text property.	Warning	Use the specified **Label ID** value for the **Text** property of a document handling button.
A document handling button on an Action Pane should have its Name property set to "Attachments".	Warning	Set the value of the **Name** property of the document handling button to **Attachments**.
All buttons on an Action Pane should have their ShowShortcut properties set to "No" to suppress the	Warning	Set the value of the **ShowShortcut** property of the action pane button control to **No**.

Message	Message type	How to fix the error or warning
addition of extra characters for mnemonic usage.		
List Page controls must not have any vertical spacing between them.	Warning	Set the value of the **VerticalSpacing** property of the control to **Auto** or **0**. Use **Auto** whenever possible.
List Pages must have their TitleDatasource property set.	Warning	Set the value of the **TitleDataSource** property in the form **Design** node to the name of a list page data source.
List Page Action Panes must have their VerticalSpacing property set to zero.	Warning	Set the value of the **VerticalSpacing** property of the **ActionPane** control to **Auto** or **0**. Use **Auto** whenever possible.

Reports Best Practice Checks

Reports are one of the central places where IntelliMorph is active. Best practices for reports concern keeping the default settings for properties.
When you design a report, you often do not know much about the environment in which the report will be executed, such as:

- The size of the paper in the user's printer.
- The length or the contents of the labels that are used in the user's installation or language.
- Which fields are disabled due to security keys and configuration keys.
- The length of the fields (extended data types) in the user's installation.
- The sort order of the data sent to the report.
- Whether the user only wants to print the (sub) totals.
- What font and size the user has set up as report defaults.
- How many records there are in the tables from which the report gets its data.

Note:
You must use labels for the report's `Caption` and `Description` properties, unless the report is country/region-specific. This enables the report to be translated more easily.
For specific rules about reports, see:

- Best Practices for Report Design
- Best Practices for Report Properties
- Best Practices for Use of Reports

Best Practices for Report Design

A report can have two kinds of design:

- Auto (AutoDesignSpecs)
- Generated (Design)

Use Auto design for all 'normal' reports.
Use a Generated design for reports with special functional requirements that cannot be implemented with Auto designs or where the design is determined externally. For example:

- Reports that are forms with externally-determined layouts where the information is expected to be placed in very specific positions.

- Reports that are forms (invoices and so on) where the design should probably be adjusted to the customer's needs at the installation. Most controls should have their positions fixed (not set to Auto) to simplify moving the controls by using the Visual Report Designer.

See Also
Best Practices for Report Properties
Reports Best Practice Checks
Best Practices for Use of Reports

Best Practices for Report Properties

This topic describes the best practices for:
- Report Properties
- Report Design Properties
- Report Section Properties
- Report Control Properties

Report Properties

Property	Rules
`Name`	The name must comply with general Naming Conventions. Prefix: Module short name Infix: Logical description of contents

Report Design Properties

Property	Rules
`Name`	Call the report 'Design', or use a logical name if you have more than one design.
`Caption`	Use a label to indicate the contents or purpose of the report. If the report is country/region-specific, you do not have to use a label.
`Description`	You must use a label for the `Description` property, unless it is a country/region-specific report.
`ReportTemplate`	Use one. In generated design reports, Report Templates are only used at generation time. Template-generated sections must be manually maintained afterwards. Use the following predefined templates: | | Internal use | External use | | List | InternalList | ExternalList | | Form | InternalForm | ExternalForm |
`Orientation`	Typically set to Auto to exploit all the IntelliMorph features. Set it to Portrait or Landscape for a report

Property	Rules
	with specific format requirements.

Report Section Properties

Property	Rules
`ColumnHeadingStrategy`	Use the default, WordWrap, if possible.

Report Control Properties

Do not change the properties of report controls for controls that are based on fields, extended data types, and display methods, with respect to

- Width
- Label
- Width of label
- Formatting information, such as the kind of decimal point, and so on

If you change these report controls, you will not be able to use IntelliMorph.

The `Left` and `Top` properties should be set to Auto on most reports, but they should have a specific value on reports that are likely to be adjusted to the needs of individual customers.

See Also

Reports Best Practice Checks
Best Practices for Use of Reports
Best Practices for Report Design

Best Practices for Use of Reports

Use of Display and Edit Methods

Security cannot be enforced on display and edit methods that are used in reports, when the method is declared on the table in the application object tree (AOT). If a display method returns data from another table (or another row or column in the same table) it can result in unwanted information disclosure. Therefore you will get a best practices error  each time you use a display or edit method in a report. You should evaluate whether each of these is a security threat, and fix them if they are, or disable the error if there is no threat. For more information, see Security on Display and Edit Methods, and "Disabling Individual Warnings and Errors" in Setting Up Best Practices Checks.

RunBaseReport

Reports should use the RunbaseReportStd framework to make it possible to run the report in batch and to give it a consistent user interface. An example can be seen in the tutorial_RunbaseReportStd report.

The menu item that is used to start the report is used to control on which tier (client, server, called from) the report should run.

Menu Item

A Menu Item for a report should be an Output Menu Item. Name the Output Menu Item after the report, or name it logically.

Consider where the report should run (Client, Server, Called), and set the property on the menu item (or the class) starting the report.

Apply the same label for the Output Menu Item, the caption of the report, and its dialog.

See Also

Best Practices for Report Properties
Reports Best Practice Checks

Best Practices for Report Design

Queries Best Practice Checks

Queries can be seen as a class interface for creating `select` statements.
Use a query in place of a `select` statement when the structure of the "select" (query) is not known until run time, or when the user should be able to specify the ranges. In other situations, use a `select` statement. Select statements are often easier to read, and they are much more compile-time stable.
Queries are embedded in Forms, Reports and RunBase jobs and should always be used with these application objects, unless there are good reasons not to.
Queries are very flexible. Everything can be specified at run time, but specify as much as possible before run time.
Create queries in the Application Object Tree (AOT) where possible, instead of building them in code.
Use intrinsic functions everywhere possible to enable compile-time checking of code.
The range of a query can be specified under program control by using two different techniques, the normal one and the query range value expression. Both take a string as an argument that is not evaluated until run time. The string must fulfill certain syntax restrictions and be as compile-time stable as possible.

Normal Query Range Values

Always use the `SysQuery::value` method when you are programmatically assigning an atomic value to a query range.
Always use the `SysQuery::range` method when you are programmatically assigning a value range to a query range. The `range` method will supply the needed ".." operators.
Always use the `SysQuery::valueEmptyString` method when you want to have a range which must have a blank value.
Always use the `SysQuery::valueUnlimited` method when you want to have a completely open range.

Query Range Value Expressions

Only use the Using Expressions in Query Ranges if the normal query range values cannot be used.
Be aware that query range value expressions are evaluated only at run time (so they are not checked at compile time).
Because the contents of the Query range value expressions should look like X++, you should be careful to format the different data types correctly. Use the `strFmt` system function or `queryValue`/`queryRange` for this. `queryValue` and `queryRange` are methods on the `Global` class.
Place the expression in a range that is defined on the most relevant of the involved fields.

See Also

X++ Standards: select Statements

Jobs Best Practice Checks

Jobs work well for testing various things. They are not intended to be shipped as a part of a solution, and it is not possible to check them in to the application object tree (AOT) version control system. (You can only use them locally on your computer.)

Menu Items Best Practice Checks

Create menu items for runnable application objects for placement in menus and as buttons on forms.

Create the menu item in the appropriate conceptual folder as follows:

- **Display** - Must be used for runnable application objects that primarily present forms, dialogs, and so on, to the user.
- **Output** - Must be used for runnable application objects whose primarily function is to print a result.
- **Action** - Must be used for runnable application objects whose primarily function is to do some kind of a job, such as creating or updating transactions in the database.

Menu Item Properties

The following table describes the best practices for menu item properties, which are in the same order that they appear in the Application Object Tree (AOT).

Property	Rules
`Name`	The `Name` property should have the same name as the object that the menu item opens.
`Label`	Use a label. Use the same label that is used for the `Caption` property of the object that the menu item opens.
`RunOn`	Ensure that you set this correctly. Client is the default value.
`ConfigurationKey`	Use the key for the module it belongs to.
`CountryConfigurationKey`	For only country/region-specific functionality, use the CSE (country/region-specific engineering) configuration key.
`WebConfigurationKey`	Use the key of the Web application it belongs to (if any).
`SecurityKey`	Mandatory unless: • The `NeededAccessLevel` property is set to NoAccess -or- • The menu item is used in the **Tools** menu. Use the security key that matches its location in the **Main** menu. For example, the **AssetBudget** menu item is used in **General Ledger** > **Inquiries.** The security key is LedgerInquiries. If a menu item is used on a button inside a form, the security key must be <module>misc.
`NeededAccessLevel`	This property sets the access level needed to start the menu item. **Warning:** If you set the `NeededAccessLevel` property to NoAccess, every user has full access to the item. Set to View for output items and most display items. A Best Practices error

Property	Rules
	occurs if you set `NeededAccessLevel` to NoAccess for output items and display items.  Set `NeededAccessLevel` to Edit, Create, or Delete for action items. For menu items that run classes, see the following section.

NeededAccessLevel for Menu Items That Run Classes

You must set the `NeededAccessLevel` property to the appropriate level for every menu item that runs a class. There are no access level settings on individual classes.
Determine the access level required for a class by using the following criteria:

- Decide what the logical function of the menu item is (the functionality from the users' point of view).
- Ascertain that when an administrator sets the permissions for a security key on a user group, the menu items appear/disappear as expected.
- Look at all the tables accessed in the class, and find out which is the "highest" access type, and then use it for the required access level:
- The "highest" access type is Delete.
- Add is lower.
- Edit is lower than Add, and so on.

Do not include "service tables," such as SysLastValue, in the analysis.

- A class that is changing a status ("Edit") for several records and is simultaneously creating transactions ("Add") to reflect that change can have the `NeededAccessLevel` set to Edit because that is the primarily logical function of the class.

Another class that has the same function, where the status change is a logical Delete, can have the `NeededAccessLevel` set to Delete. This is partially checked by the `checkMenuItemActionAccessLevel` method in the `SysApplCheck` class.

Web Best Practice Checks

When you create HTML documents or contents from X++ code, use the following standards.

Formatting Strings with Web Tags

The `Web` class has a set of functions to format and output strings with Web tags. Use these, if possible.
To build strings:

1. Construct the user interface string by using the `strFmt` system function.
2. Insert HTML tags.

For example:

```
strFmt('<h2>%1</h2>',strFmt("@SYS4711",accountNum))
```

Where to Place the Code

Put logic in classes (and tables), so it can be reused in a new context, like a Web page.

Transactions

Keep locks on shared records to an absolute minimum. Use as few locks as possible, and for as short time as possible.

Transactions should be started only after the user has pressed a button, and they be completed before the control is returned to the user (the user may navigate to another Web page, and forget about the transaction on the current page).

Workflow Best Practice Checks

Best practice checks for workflow focus on the properties of workflow categories, templates, approvals, and tasks. The checks verify that the properties are correctly configured for execution. For information about how to set the options for best practice checks, see Best Practice Options. This topic includes general principles to follow when you develop workflows and a table of the best practice checks and how to fix the errors.

General Principles

When creating a query for a workflow document, consider the following:

- Most document data is used to set up conditions for workflow. In addition, the query identifies workflow placeholders used in messages in the configuration user interface for instructions and notifications. Make sure to use placeholder fields that the end-user can understand and not fields such as `RecId` and infrastructure fields.
- By default, all data fields with the `Visible` property in the table set to Yes are exposed for workflow conditions. To simplify the set up for workflow conditions, you can remove unnecessary data fields that will not be used for workflow conditions or placeholders in the configuration UI. To remove data fields, set the `Dynamic` property on the **Fields** node of the data source to No. Expand the **Fields** node, right-click a field, and then click **Delete**. If the query is not saved, the **Delete** command is not available.
- Set the `Relations` property of the query to Yes if the query system should use the relations that are defined for tables and extended data types. When set to Yes, the query is automatically updated if a relation is changed.
- Optionally, set the `Join Mode` property of the query to Outer Join. Records in one data source are joined even if there are no matching values in the joined field from the second data source. Only fields from `Inner Join` or `Outer Join` are visible in the configuration user interface as conditions and placeholders.
- In the **Properties** sheet, set the `Table` property to the table that contains the data for the workflow document. The root table in the query should be the same as the workflow data source property of the form.

When creating task and approval elements in the Application Object Tree (AOT), be sure to use unique and declarative label names for the element. When adding an approval to a workflow configuration, the label **Approval** would not be unique enough for the user to determine which approval is to be added.

Use the WorkflowWorkItemActionManager Class to manage task or approval outcome action menu items for type `Complete`, `Return`, `RequestChange`, and `Deny`.

Best Practice Checks

The following table lists the best practices error messages and how to fix the errors.

Message	Message type	How to fix the error or warning
Configuration key not defined	Error	The workflow template or a workflow element `ConfigurationKey` property setting is invalid or not defined. Set the `ConfigurationKey` property to a valid setting. This property is optional.
Workflow document not defined	Error	The workflow template, task, or approval `Document` property setting is invalid or not

Message	Message type	How to fix the error or warning
		defined. Set the `Document` property to a valid setting. For more information, see How to: Create a Workflow Document Class. This property is required.
Workflow document does not reference a valid class deriving from the WorkflowDocument Class	Error	The workflow template, task, or approval `Document` property setting must reference a class that extends the WorkflowDocument Class and overrides the WorkflowDocument.getQueryName Method. The class was probably deleted, renamed, or no longer extends the WorkflowDocument Class. For more information, see How to: Create a Workflow Document Class.
Action menu item not defined	Error	The task or approval `ResubmitMenuItem` or `DelegateMenuItem` property setting is invalid or not defined. Set the menu item to a valid setting. For more information, see How to: Associate an Action Menu Item with a Workflow Task or Approval Outcome. This property is optional.
Reference to action menu item is invalid	Error	The workflow template, task, or approval menu item property setting is invalid or no longer exists. The action menu item was probably deleted or renamed. For more information, see Creating a Workflow Template.
Reference to Web action menu item is invalid	Error	The workflow template, task, or approval Web menu item property setting is invalid or no longer exists. The Web action menu item was probably deleted or renamed. For more information, see Creating a Workflow Template.
Display menu item not defined	Error	The task or approval display `DocumentMenuItem` property setting is invalid or not defined. Set the menu item to a valid setting. For more information, see How to: Associate a Display Menu item with a Workflow Task or Approval. This property is required.
Reference to display menu item is invalid	Error	The task or approval display `DocumentMenuItem` property setting is invalid or no longer exists. The display menu item was probably deleted or renamed. For more information, see How to: Create a Workflow Task and How to: Create a Workflow Approval.
Reference to Web URL menu item is invalid	Error	The task or approval display `DocumentWebMenuItem` property setting is invalid or no longer exists. The display menu item was probably deleted or

Message	Message type	How to fix the error or warning
		renamed. For more information, see How to: Create a Workflow Task and How to: Create a Workflow Approval.
Event handler should be defined	Warning	The task or approval `EventHandler` property is invalid or not defined. Set the `EventHandler` property to the event handler for that workflow element. For more information, see Handling Workflow Events. This property setting is optional. However, when an event handler is not defined, no application logic will run when this event is triggered.
Event handler not defined	Error	The task or approval outcome `EventHandler` property is not valid or defined. Set the `EventHandler` property to the event handler for that workflow element. For more information, see Handling Workflow Events. This property is required for all outcomes that are enabled.
Event handler does not reference a valid class implementing the '%1' interface	Error	All workflow template event handlers must implement the appropriate workflow interface. For example, the WorkflowCompletedEventHandler Interface must be implemented by the class that is referenced by the workflow template `CompletedEventHandler` property setting. For more information, see Handling Workflow Events.
Module not defined	Warning	The workflow category `Module` property setting refers to a module that is invalid or no longer exists. The module was probably deleted or renamed. For more information, see ModuleAxapta Enumeration. This property is required.
Category not defined	Error	The workflow template `Category` property setting is not defined. Set the workflow template `Category` property setting to a valid category. This property is required.
Invalid reference to workflow category	Error	The workflow template `Category` property setting is invalid or no longer exists. The category was probably deleted or renamed. For more information, see How to: Create Workflow Categories.
Required element '%1' does not reference a valid workflow element	Error	The specified task or approval workflow element in the **Required Elements** node property setting is invalid or no longer exists. The workflow element was probably renamed or deleted. For more information, see How to: Add Required Tasks and Approvals to a Workflow

Message	Message type	How to fix the error or warning
		Template.
Required element '%1' does not reference same document as the workflow template	Error	The `Document` property setting of the specified task or approval workflow element in the workflow template **Required Elements** node does not match the `Document` property setting of the workflow template. Set the specified workflow element `Document` property setting to match the workflow template `Document` property setting. For more information, see How to: Create a Workflow Task and How to: Create a Workflow Approval.
Participant provider does not reference a valid class implementing the WorkflowParticipantProvider Interface	Error	The task or approval `ParticipantProvider` property setting must reference a participant provider class that implements the WorkflowParticipantProvider Interface. For more information, see Defining Workflow Providers.
Hierarchy provider does not reference a valid classimplementing the WorkflowHierarchyProvider Interface	Error	The task or approval `HierarchyProvider` property setting must reference a hierarchy provider class that implements the WorkflowParticipantProvider Interface. For more information, see Defining Workflow Providers.
One of the properties `ParticipantProvider` or `HierarchyProvider` must be defined	Error	The task or approval `HierarchyProvider` or `ParticipantProvider` property setting is invalid or not defined. Set the `HierarchyProvider` or `ParticipantProvider` property to a valid setting. For more information, see How to: Add a Workflow Provider to a Task or Approval. Either the `HierarchyProvider` or `ParticipantProvider` property is required. Optionally, both the `HierarchyProvider` and `ParticipantProvider` property may be set.
Due date provider not defined	Error	The task or approval `DueDateProvider` property setting is invalid or not defined. Set the `DueDateProvider` property to a valid setting. For more information, see How to: Add a Workflow Provider to a Task or Approval. This property is required.
Due date provider does not reference a valid class implementing the WorkflowDueDateProvider Interface	Error	The task and approval `DueDateProvider` property setting must reference a due date provider class that implements the WorkflowDueDateProvider Interface. For more information, see Defining Workflow Providers.
Approve outcome must exist and be enabled	Error	The **Approve** outcome in the approval **Outcomes** node has the `Enabled` property

Message	Message type	How to fix the error or warning
		setting set to No. Each approval must have at least one enabled outcome of type `Complete` and an action menu item for the approval work item button to display in the user interface. For more information, see How to: Create a Workflow Approval.
Element outcome '%1' `ActionMenuItem` property not defined	Error	The specified task or approval outcome action menu item is invalid or not defined. Set the action menu item to a valid setting. For more information, see How to: Associate an Action Menu Item with a Workflow Task or Approval Outcome. This property is optional.
Element outcome '%1' `ActionWebMenuItem` property does not reference a valid Web action menu item	Error	The specified task or approval outcome action Web menu item is invalid or no longer exists. The specified action Web menu item was probably deleted or renamed. For more information, see How to: Create a Workflow Task and How to: Create a Workflow Approval.
Element outcome '%1' `ActionMenuItem` property does not reference a valid action menu item	Error	The specified task or approval outcome action menu item is invalid or no longer exists. The specified action menu item was probably deleted or renamed. For more information, see How to: Create a Workflow Task and How to: Create a Workflow Approval.
Element outcome '%1' `EventHandler` property should be defined	Warning	The specified task or approval event handler is invalid or not defined. Set the event handler to a valid setting. For more information, see Handling Workflow Events. This property is optional. However, when an event handler is not defined, no application logic will run when this event is triggered.
Element outcome '%1' `EventHandler` property does not reference a valid class implementing the '%2' interface	Error	The specified task and approval outcome event handler property setting must reference a class that implements the associate event handler interface. An approval, for example, has an **Approve** outcome of type `Complete` and the `EventHandler` property setting must reference a class that implements the WorkflowElementCompletedEventHandler Interface. For more information, see Handling Workflow Events.
Task outcomes must contain one enabled outcome of type `Complete`	Error	There are no enabled outcomes in the task **Outcomes** node of type `Complete`. Each task must have at least one enabled outcome of type `Complete` and an

Message	Message type	How to fix the error or warning
		action menu item for the task work item button to display in the user interface. For more information, see How to: Create a Workflow Task.

X++ Coding Standards

General principles are:

- Declare variables as locally as possible.
- Check the error conditions in the beginning; return/abort as early as possible.
- Have only one successful return point in the code (typically, the last statement), with the exception of `switch` cases, or when checking for start conditions.
- Keep the building blocks (methods) small and clear. A method should do a single, well-defined job. It should therefore be easy to name a method.
- Put braces around every block of statements, even if there is only one statement in the block.
- Put comments in your code, telling others what the code is supposed to do, and what the parameters are used for.
- Do not assign values to, or manipulate, actual parameters that are "supplied" by value. You should always be able to trust that the value of such a parameter is the one initially supplied. Treat such parameters as constants.
- Clean up your code; delete unused variables, methods and classes.
- Never let the user experience a runtime error. Take appropriate actions to either manage the situation programmatically or throw an error informing the user in the **Infolog** about the problem and what actions can be taken to fix the problem.
- Never make assignments to the "this" variable.
- Avoid dead code. (See Dead Code Examples.)
- Reuse code. Avoid using the same lines of code in numerous places. Consider moving them to a method instead.
- Never use `infolog.add` directly. Use the indirection methods: `error`, `warning`, `info` and `checkFailed`.
- Design your application to avoid Deadlocks.

More specific code standards are described below:

- X++ layout
- Comments
- Semicolons
- Constants
- Arrays
- Dates
- try/catch statements
- throw statements
- ttsBegin and ttsCommit
- if ... else and switch statements
- select Statements

X++ Layout

General Guidelines

- Only one statement per line.
- Break up complex expressions that are more than one line - make it visually clear.
- Use a single blank line to separate entities.
- Do not use parentheses around the case constants.
- Do not use parentheses around `where` expressions.
- Add one space between `if`, `switch`, `for`, `while` and the expressions starting parentheses. For example:

```
if (creditNote)
```

- Use braces around all code blocks, except for around case clauses in a `switch` statement. Use braces even if there is only one statement in the block.

Indentation

An indentation is equivalent to four (4) characters, which corresponds to one tab in the X++ editor. You must not start a new line in columns 2, 3 or 4.

- Put opening and closing braces, { and }, on the same level, on separate lines, and aligned with the command creating the block. They must be at the start of a line, and in a tab column position (1, 5, 9 etc.). Braces must be on a dedicated line unless a opening brace is followed by a semicolon (`{;`) or a closing brace is followed by a `while` (`}while`).
- The following reserved words should be placed at the beginning of a line: `case`, `catch`, `changeCompany`, `continue`, `default`, `else`, `for`, `if`, `retry`, `return`, `switch`, `try`, `ttsAbort`, `ttsBegin`, `ttsCommit`, `while`.
  The exceptions to this rule are:
  `case: …` (reserved words in a `case` statement)
  `default: …` (reserved words in a `default` statement)
  `else if`
  `}while`
- If a line of code starts with any other reserved word, or with an alphabetical character, the line should start in a tab column position (1, 5, 9 etc). The following reserved words must be placed in a tab column position: `case`, `catch`, `changeCompany`, `continue`, `default`, `else`, `for`, `if`, `retry`, `return`, `switch`, `try`, `ttsAbort`, `ttsBegin`, `ttsCommit`, `while`.
  The exceptions to these rules are:
  `case: …` (reserved words in a `case` statement)
  `default: …` (reserved words in a `default` statement)
  `else if`
  `}while`
- The reserved word `else` must have a dedicated line unless you write `else if`.
- `switch-case` statements: indent case and default statements by 1 level (with any code within these indented a further level) and indent break statements by two levels.
- Indent `where` and other qualifiers to the select statement by one level.
- If Booleans are used in a long expression, put them at the start of an indented new line.

Example switch-case Statement

```
switch (myEnum)
{
    case ABC::A:
        ...
        break;
    case ABC::B
        ...
        break;
    default:
        ...
        break;
}
```

Example select Statement

```
select myTable
    index hint myIndex
    where myTable.field1 == 1
        && myTable.field2 == 2;
```

Example Layout of Booleans in a Long Expression

```
select firstOnly utilElements
    where utilElements.recordType == recordType
        && utilElements.parentId == parentID
        && utilElements.name == elementName;
```

Column Layout

Column layout should be used where more than one line has the same structure; for example, in declarations or assignments.
Do not use column layout when there is only one row, or if consecutive rows do not have the same structure.
Column format is defined using extra blanks.

Example Column Layout

```
tmpABC.refRecId     = inventTable.recId;
tmpABC.itemGroupId  = inventTable.itemGroupId;
tmpABC.itemId       = inventTable.itemId;
tmpABC.amount       = amount;
tmpABC.oldValue     = this.getCategory(inventTable);
tmpABC write();
```

Layout for Methods

The starting parenthesis on method declarations and calls should be the character just after the method name (no space).
If there are one or two parameters, the parameters can be listed on the same line. If there are more than two parameters, move each parameter onto a new line, and indent by 4 spaces.

Example Layout for Method with One or Two Parameters

```
myMethod(parameter1, parameter2);
```

Example Layout for Method with Many Parameters

```
myMethod(
    parameter1,
    parameter2,
    parameter3);
```

See Also

X++ Coding Standards

X++ Standards: Comments

Use // for both single and multiline (block) comments. There should be a space between the "//" and the start of the comment.
Comments should be in US-English and start with an uppercase letter (unless the first word is a lowercase name).
Put comments on a separate line before the code they are describing. The only exception to this is when you are describing parameters. In this case, put one parameter per line, with the comment describing it to the right of the parameter.
When creating a multiline comment, do not write on the first or the last line of the comment (as shown in the following example).
For example:

```
// Single line comment
```

```
<code>

//
// Comment on multiple lines.
// Do not add text to 1st or last line of comment.
//
<code>
```

Comments should not include:

- Dates
- Names
- Aliases
- Version or layer references
- Bug numbers – unless it is a workaround, or unless the code could appear inappropriate if you didn't know that it was for a bug fix.
- Politically or culturally sensitive phrases

Note:
If you put a comment at the start of the method to describe its purpose and use, you can use block comments (/* */).

Remove Commented-out Code from the Application

The Best Practice is that we don't ship a product with commented-out code, so any commented-out code should be removed.

Tip:
To find comments in the source (both // .. and /* .. */), use the **Find** dialog to search for methods containing the text (regular expression): /[/*]

See Also
X++ Coding Standards

X++ Standards: Using Semicolons

Always place a semicolon (;) on an empty line in front of the first statement in your code (after the variable declarations).
This is particularly important if the statement does not begin with a keyword (`select`, `while`, and so on). For example, the first part of the statement is a variable or a type reference. You should use a semicolon even if the code compiles. Types introduced later (that have the same name as the first part of the statement) might prevent the code from compiling.

See Also
X++ Coding Standards

X++ Standards: Constants

Follow the best practices rules about using constants. These are designed to make it easier to maintain X++ code.

Constants

Rule	Error level
Do not use hard-coded constants (except 0, 1, 100).	Warning
Define constants in a method, class declaration, or if necessary globally in a	None

Rule	Error level
macro. Reuse existing constants. Consider alternative ways of getting the constant: • Intrinsic Functions • `maxInt`, `minInt`, `funcName`, `maxDate` system functions • Global macros	

User Interface Text

Rule	Error level
User interface text must be in double quotes, and you must always use a label (also in double quotes).	Error
User interface labels must be complete sentences. Do not build sentences using more than one label, or other constants or variables under program control (do not use concatenation). Example: `Description description = "@SYS12345"` Use `strFmt` to format user interface text.	None

System-oriented Text

Rule	Error level
System-oriented text constants must be in single quotes.	None
Do not hard-code text.	Warning

Do not use labels. You will get a warning if a label is used inside single quotes.
Example:
`#define.Filename('myFile.txt')`
`Filename filename = #Filename;`
	Warning

Numeric Constants

Rule	Error level
Always review the direct use of numeric constants, except for 0 meaning null, 1 meaning increment, and 100 when calculating percents and currencies.	None
Certain numeric constants are predefined, such as the number of days per week, and the number of hours per day. For example, see the `TimeConstants` and `SysBitPos` macros in the Application Object Tree (AOT).	None

See Also
X++ Coding Standards

X++ Standards: Arrays

This topic describes the best practice for using the memory option in arrays. For more information, see Arrays.
The memory option is the optional second array declaration option. It specifies how many consecutive entries in the array will be held in memory at a particular time. The rest will reside on disk (a cached temporary file, indexed by the array index).
Dynamic arrays are sized according to the maximum index used.

Tip:
Use the memory option to limit the amount of RAM used to hold the data of the array when you work with high numbered indexes (and large cells).
If you use all (or nearly all) of the entries in an array, set the memory option to a large number, or do not set it at all.
If you only use a few of the entries in the array, set the memory option to a small number, such as 1.

Read Performance

If you consider using the memory option on an array where you use all (or almost all) of the entries, the look up performance should be considered.
If you traverse an array sequentially, such as with an index of 1, 2, 3, ..., n, you will probably not experience any read performance problems. The cell data blocks will be read sequentially from disk and they will be read to the end before the next block is read (disk reads will be number of entries/memory option size).
If you traverse an array randomly (such as with an index of 300, 20, 5, 250, n, ..., 50) the cell data will also be read from disk randomly, so you may experience read performance problems (disk reads could be as high as the number of entries).

Example 1

```
MyTable myTable;
boolean foundRecord[,1];
;
while select myTable
    where myTable
    ...
{
    foundRecord[myTable.recId] = true;
    ...
}
```

Example 2

```
CustTable   custTable;
CustAccount foundAccount[];
int         i;
;
while select custTable
    where custTable
    ...
{
    i++;
    foundAccount[i] = custTable.AccountNum;
    ...
}
```

Example 3

```
Name foundName[,100];
```

```
int  i;
;
while select custTable
    where custTable
    ...
{
    i++;
    foundName[i] = custTable.Name;
}
```

See Also
X++ Coding Standards
Swapping Arrays to Disk

X++ Standards: Dates

When you are programming with dates, Best Practices are:

- Use only strongly typed (date) fields, variables, and controls (do not use `str` or `int`).
- Use Auto settings in date formatting properties.
- Use `DateTimeUtil::getSystemDateTime` instead of `systemDateGet` or `today`. The `today` function uses the date of the machine. The `systemDateGet` method uses the system date in Microsoft Dynamics AX. Only `DateTimeUtil::getSystemDateTime` compensates for the time zone of the user.
- Avoid using `date2str` for performing date conversions.

Use Strong Typing (date)

Never permanently store a date in anything other than a date field.
Always present a date in a date control.
Fields, variables, and controls should be defined by extended data types. These should have their formatting properties set to **Auto** so that you do not take control away from the users' specific date formatting setup.

Current Business Date

Most application logic should use the system function `systemDateGet`, which holds the logic business date of the system (this can be set from the status bar).
The system function `today()` should be used only where the actual machine date is needed. This is seldom the case.

Note:
The date and time can be different on the client and the server.

Avoid String / Date Conversions

You will not typically need to format a date to a string. Use date fields, variables, and date controls on forms and reports instead.
If you need to format a date to a string:

- For user interface situations, use `strFmt` or `date2Str` with -1 in all the formatting parameters. This ensures that the date is formatted in the way that the user has specified in **Regional Settings**.
- For other specific system-related situations, such as communication with external systems, use `date2Str`.

When you let **Regional Settings** dictate the format, be aware that it can change from user to user and might not be a suitable format for external communication.
Using `str2Date` indicates that dates are being used that have had a string format.

X++ Standards: try/catch Statements

Always create a try/catch deadlock/retry loop around database transactions that might lead to deadlocks.
Whenever you have a retry, all the transient variables must be set back to the value they had just before the try. The persistent variables (that is, the database and the **Infolog**) are set back automatically by the throw that leads to the catch/retry.

Example

```
try
{
    this.createJournal();
    this.printPosted();
}
catch (Exception::Deadlock)
{
    this.removeJournalFromList();
    retry;
}
```

See Also
X++ Coding Standards

X++ Standards: throw Statements

The `throw` statement automatically initiates a `ttsAbort`, which is a database transaction rollback.
The `throw` statement should be used only if a piece of code cannot do what it is expected to do. The `throw` statement should not be used for more ordinary program flow control.
Always place an explanation of the throw in the **Infolog** before the actual throw.

Note:
Do not use `ttsAbort` directly; use `throw` instead.

See Also
X++ Coding Standards

X++ Standards: ttsBegin and ttsCommit

`ttsBegin` and `ttsCommit` must always be used in a clear and well-balanced manner. Balanced `ttsBegin` and `ttsCommit` statements are the following:

- Always in the same method.
- Always on the same level in the code.

Avoid making only one of them conditional.
Use `throw`, if a transaction cannot be completed.
Do not use `ttsAbort`; use throw instead.
Do not use anything that requires a user interaction within a transaction (such as an action on a dialog box).

See Also
X++ Coding Standards

X++ Standards: if … else and switch Statements

This topic describes X++ code style standards for the `if...else` statement and the `switch` statement.

if...else

If you have an `if...else` construction, then use positive logic:
Preferred:

```
if (true)
{
    ...
}
else
{
    ...
}
```

Avoid:

```
if (!false)
{
    ...
}
else
{
    ...
}
```

It is acceptable to use negative logic if throwing an error, and in cases where the use of positive logic would make the code difficult to understand.
There should be a space character between the if keyword and the open parenthesis.

Switch Statements

Always end a case with a break statement (or return/throw). If you intentionally want to make use of the fall-through mechanism supported in X++, replace the missing break statement with a comment line:

```
    // Fall through
```

This comment line makes it visually clear to the reader that the fall-through mechanism is utilized.
Use 3 levels of indentation:

```
switch (Expression)
{
    case: Constant:
        Statement;
        break;
    ...
}
```

Do not put parentheses around cases. 
There should not be a space between the case keyword and the colon character.

Use a switch Instead of Nested if ... else Statements

Use `switch` statements instead of nested `if...else` statements.
Recommended:

```
switch (myEnum)
{
    case ABC::A:
    ...
        break;
    case ABC::B:
    ...
        break;
    case ABC::C:
    ...
        break;
    default:
    ...
        break;
}
```

Avoid:

```
if (myEnum == ABC::A)
{
    ...
}
else
{
    if (myEnum == ABC::B)
    {
        ...
    }
    else
    {
        if (myEnum == ABC::C)
        {
            ...
        }
        else
        {
           ...
        }
    }
}
```

X++ Standards: select Statements

The basic rules for `select` statements are as follows:

- Specify what records you need by using the `where` statement to limit the returned rows. To sort the returned records, use the `order by` statement.
- Add an index hint on realistic data only if you discover performance problems without the hint.
- Use the `index` keyword if the order should be defined by the index definition.

Ordering of Data

Most `select` statements should be written without an index or an index hint, leaving the job of ordering the data to the optimizer in the database system. Whenever you use the index hint functionality, make a comment about why you are explicitly specifying it, thereby taking control away from the database management system (DBMS). Consider using the `forceLiterals` or `forcePlaceholder` statements as well or instead of `select` statements.

Note:

Be careful when using the forceLiterals keyword in X++ `select` statements. It could expose code to an SQL injection security threat.
Use the `order by` statement when you want the data ordered, and you do not want to specify which index to use. Ordering of data can take time and should only be done if needed.
Use the `index` keyword when you want the data to be centrally ordered as specified in the index specification on the table. If the index is specified as Enabled: No, an index is not generated by the database system.
The use of the `index` statement can sometimes result in an index hint.

Using the firstOnly Qualifier

If you are going to use only the first record or if only one record can be found, use the `firstOnly` qualifier. This optimizes the `select` statement for only one record.

Note:

The `firstOnly` qualifier does not guarantee that only one record is returned.
Do not use `while select firstOnly`.
The `firstOnly` qualifier and the field list are implicit in `(select...).`<*field*> statements, and are not explicitly needed. Use this kind of `select` statement wherever reasonable.
It is a best practice to use the `firstOnly` qualifier in `find` methods on tables.

Using select Statements Locally in Methods

If a `select` statement is local to a method, use a field list to increase performance. If you use a `select` or a `while select` statement and the size of the fields that are used total less than 50 percent of the total record size, a warning appears if you do not use a field list.
If a `select` statement is local to a method and the `firstOnly` qualifier is not used, you must use the `next` statement on the result set.

Layout Examples

Here are two (contrived) examples of how to find ledger transactions in account number, transaction date order.
The following code shows how to find the last week's (that is, a few days) transactions on all of the (many) Profit and Loss accounts.

```
select firstOnly ledgerTrans
    index hint DateIdx
    order by accountNum, transDate
    where ledgerTrans.accountNum >= '40000'
        && ledgerTrans.accountNum <= '99999'
        && ledgerTrans.transDate  >= 26\04\1999
        && ledgerTrans.transDate  <= 02\05\1999;
```

The following code shows how to find transactions for the entire year (many dates) on (the few) liquid assets accounts.

```
while select ledgerTrans
    order by accountNum, transDate
    where ledgerTrans.accountNum  >= '11100'
        &&  ledgerTrans.accountNum <= '11190'
        &&  ledgerTrans.transDate  >= 01\07\1999
        &&  ledgerTrans.transDate  <= 30\06\2000
{
    // Do whatever is needed.
    print ledgerTrans.amountMST;
}
```

Following are best practices from the previous examples:

- The `index`, `order by`, and `where` statements are indented once relative to the `select` or `while select` statements.
- The `where` expression is structured in a column.
- The Boolean operators (&&) are at the beginning of the line (and in columns).
- The `while select` block has braces even though it contains only one statement.
- The braces are at the same column position as the `while` block.
- The uppercase- and lowercase-name standards are adhered to.

See Also
Queries Best Practice Checks

Intrinsic Functions

Intrinsic functions are metadata assertion functions. They take arguments that represent entities in the Application Object Tree (AOT), and validate these arguments at compile time. They have no effect at run time. Intrinsic functions are a subgroup of the Functions, and typically have names ending in Num or Str, for example: `classNum` and `formStr`.
Intrinsic functions should be used wherever possible in X++ code to make the code resilient to changes to the metadata stored in the AOT.
You will get a best practice warning if you use the for `identifierStr` function. This is because no existence checking is carried out for `identifierStr`. Try to use a more specific intrinsic function if one is available.
The following list contains the intrinsic functions in MorphX.

Validation of Table Metadata

- tableCollectionStr
- tableFieldGroupStr
- tablePName
- tableNum
- tableMethodStr
- tableStaticMethodStr
- tableStr

Validation of Field Metadata

- fieldNum
- fieldPname
- fieldStr

Validation of Index Metadata

- indexNum
- indexStr

Validation of Data Type Metadata

- enumCnt
- enumNum
- enumStr
- extendedTypeNum
- extendedTypeStr
- typeId

Validation of Configuration and Security Key Metadata

- configurationKeyNum
- configurationKeyStr
- securityKeyNum
- securityKeyStr

Validation of License Metadata

- licenseCodeNum
- licenseCodeStr

Validation of Class Metadata

- classNum
- classStr
- methodStr
- staticMethodStr

Validation of Form, Report, Query, and Menu Metadata

- formStr

Note:

In forms, `control::`*controlName* returns the ID of the control. Where possible, use the `AutoDeclaration` property on controls.

- menuItemActionStr
- menuItemDisplayStr
- menuItemOutputStr

- menuStr
- reportStr
- queryStr

Validation of Web Metadata

- webActionItemStr
- webDisplayContentItemStr
- webletItemStr
- webOutputContentItemStr
- webpageDefStr
- webReportStr
- websiteDefStr
- websiteTempStr
- webStaticFileStr
- webUrlItemStr
- webWebpartStr

Other

- identifierStr
- literalStr
- maxDate
- maxInt
- minDate
- minInt
- resourceStr
- varStr

Clear Code Examples

The code in the two examples below can be rewritten to be much clearer.

Example 1

From:

```
if (args.parmEnumType() != enumnum(BMBuildIdEnum))
{
    if (args.record().tableId == tableNum(BMmoduleTable))
    {
        moduleTable = args.record();
        buildId = moduleTable.buildId;
    }
    else
    {
        return null;
    }
}
else
{
    buildId = args.parmEnum();
}
...
```

To:

```
if (args.parmEnumType() == enumNum(BMBuildIdEnum))
{
    buildId = args.parmEnum();
}
else
{
    if (args.record().tableId == tableNum(BMmoduleTable))
    {
```

```
        moduleTable = args.record();
        buildId = moduleTable.buildId;
    }
    else
    {
        return null;
    }
}
...
```

The rewrite puts the most important case first, and removes the negative logic used in the first `if` statement in the first version of the code.

Example 2

From:

```
ledgerJournalTrans = this.ledgerJournalTransInitFromCreate(_tmpProjAdjustmentCreate);
    if (ledgerJournalTrans.validateWrite())
    {
        ledgerJournalTrans.insert();
        ProjPostLedger = ProjPost::construct(ledgerJournalTrans,ledgerVoucherTrans);
        if (projPostLedger.checkTrans())
        {
            projPostLedger.PostTrans();
        }
        else
        {
            throw error("@SYS21628");
        }
    }
    else
    {
        throw error("@SYS21628");
    }
    ledgerjournalTrans.delete(false);
...
```

To:

```
ledgerJournalTrans = this.ledgerJournalTransInitFromCreate(_tmpProjAdjustmentCreate);
if (!ledgerJournalTrans.validateWrite())
{
    throw error("@SYS21628");
}
ledgerJournalTrans.insert();
ProjPostLedger = ProjPost::construct(ledgerJournalTrans,ledgerVoucherTrans);

if (!projPostLedger.checkTrans())
{
    throw error("@SYS21628");
}
    projPostLedger.PostTrans();
    ledgerjournalTrans.delete(false);
...
```

See Also

Dead Code Examples
X++ Coding Standards

Dead Code Examples

Avoid writing redundant code.

Example 1

In the following example, `b++` is never reached:

```
a++;
return a;
b++;
return b;
```

Example 2

In the following example, the break statement is never reached:

```
switch (type)
{
    case UtilElementType::Job:
         return false;
         break;
    ...
```

Example 3

In the following example, return `a` is never reached:

```
if (!a)
{
    throw error("@SYS21628");
    return a;
}
b++;
return b;
```

Example 4

In the following example, the `else` statement is never used, because execution has already ended at the return statement:

```
if (a)
{
    return a;
}
else
{
    b++;
    return b;
}
```

Use this format instead:

```
if (a)
{
    return a;
}
b++;
return b;
```

See Also

Clear Code Examples
X++ Coding Standards

Best Practices: XML Documentation

Microsoft Dynamics AX conducts a best practices check of the XML comments to make sure that you provide documentation in the appropriate tags. For information about how to set the options for best practice checks, see Best Practice Options.

Best Practice Checks

The following table lists the best practices error messages and how to fix the errors.

Message	Message type	How to fix the error or warning
Tag '%1' in XML documentation is not supported.	Warning	Add XML documentation. For information about how to add XML documentation, see How to: Add XML Documentation to X++ Source Code. Because this is a warning instead of an

Message	Message type	How to fix the error or warning
		error, this is optional.
Unsupported tag '%1' in XML documentation.	Error or Warning	Check the casing of the XML tags if this is reported as an error. If this is reported as a warning, an unsupported tag is present. Remove unsupported tags.
Missing tag '<param name="%1">' in XML documentation.	Error	Add <param> tags to the XML header template. The <param> tag must have a name attribute. The value of the attribute is case-sensitive and must match the casing in the method.
Missing tag 'returns' in XML documentation.	Error	Add <returns> tags to the XML header template.
Missing tag 'summary' in XML documentation.	Error	Add <summary> tags to the XML header template.
Tag '%1' exists more than once in XML documentation.	Error	Delete extra tags. This applies only when multiple tags are not appropriate.
Tag '<param name="%1">' has no content in XML documentation.	Error	Add a description of the parameter between the <param> tags.
Tag '<param name="%1">' in XML documentation doesn't match actual implementation.	Error	Fix the value of the name attribute. It is case-sensitive and must match the casing in the method.
Tag 'exception' has no content in XML documentation.	Error	Add a description of the exception between the <exception> tags.
Tag 'permission' has no content in XML documentation.	Error	Add a description of the required permission between the <permission> tags.
Tag 'remarks' has no content in XML documentation.	Error	Add content between the <remarks> tags or delete the <remarks> tags.
Tag 'returns' has no content in XML documentation.	Error	Add a description of the return value between the <returns> tags.
Tag 'returns' in XML documentation doesn't match actual implementation.	Error	Delete the <returns> tags and the description of the return value.
Tag 'summary' has no content in XML documentation.	Error	Add a topic summary between the <summary> tags.
XML documentation is not well-formed.	Error	Make sure that there are no mistakes in the XML tags. Each opening tag must have a corresponding closing tag.
Tag 'seealso' has no content in	Error	Add content between the

Message	Message type	How to fix the error or warning
XML documentation.		<seealso> tags or delete the <seealso> tags.
No XML documentation for this method.	Error	XML documentation has not been written for this method.

Example

Description

The following example shows the XML header template with documentation for the `DocuAction.newArgs` method.

Code

```
/// <summary>
/// Creates a new instance of the DocuAction class.
/// </summary>
/// <param name="args">
/// The arguments that are used to create the DocuAction instance.
/// </param>
/// <returns>
/// The new DocuAction instance.
/// </returns>
/// <exception cref="Exception::Error">
/// A valid DocuAction object could not be created.
/// </exception>
static DocuAction newArgs(Args args)
{
    FormDataSource              formDataSource;
    DocuRef                     docuRef;
    FormFunctionButtonControl   ctrl;
    Object                      formRun = args.caller();
    DocuType                    docuType;
    DocuAction                  createdDocuAction;
    ;

    ctrl            = formRun.selectedControl();

    docuType = DocuType::find(ctrl.name());
    if (!docuType || !docuType.verifyParameters(true))
        return null;

    formDataSource  = formRun.datasource();
    formDataSource.create();
    docuRef         = formDataSource.cursor();
    docuRef.TypeId  = ctrl.name();
    formDataSource.write();

    formDataSource.refresh();

    createdDocuAction = DocuAction::newDocuRef(docuRef);
    if (createdDocuAction != null)
```

```
    {
        createdDocuAction.verify(true);
    }
    return createdDocuAction;
}
```

Best Practices: Avoiding Potential Security Issues

Some of the X++ APIs may have potential security issues. For example, they might allow unauthorized access to the database or the Application Object Tree (AOT), if used in a nonsecure manner.

If a call to one of these potentially unsafe APIs generates a Best Practices error, this indicates that you should assess the security implications of using the method. You may need to apply Code Access Security by using one of the classes derived from CodeAccessPermission Class, and/or take other mitigating actions, such as validating user input.

When you are satisfied that the security implications of using the class have been investigated and mitigated, you can turn off the best practice error by adding the following comment above the call to the method.

```
// BP Deviation documented
```

There is more information about the mitigations for each potentially unsafe API in the Help topics for the classes you received the error message for.

For more information about the APIs protected by Code Access Security, see Secured APIs.

Microsoft Dynamics AX conducts a best practices check of the XML comments to be sure that you provide documentation in the appropriate tags. For information about how to set the options for best practice checks, see Best Practice Options.

Best Practice Checks

The following table lists the best practices error messages and how to fix the errors.

Message	Message type	How to fix the error or warning
TwC: Validate data displayed in form is fetched using record level security. Dangerous API %1 used.	Error	Assess the security implications of using the method. You may need to apply Code Access Security by using one of the classes derived from CodeAccessPermission Class. For information about record level security, see Record Level Security. For more information about security, see Writing Secure X++ Code.

See Also

Setting Up Best Practices Checks

Naming Conventions

Naming conventions contribute to consistency and to making the application easier to understand.
There are specific naming rules for the following application objects:

- Table Properties
- Table Field Properties
- Tables
- Views
- Extended Data Types
- Enums
- Security keys
- Classes
- Interfaces
- Methods
- Parameters
- Forms
- Lookup Forms
- Form Data Sources
- Reports
- Indexes
- Variables
- License codes

General Rules

- All names must be in U.S. English.
- The default rule is to use logical and descriptive names if no other specialized rules apply.
- Identifier names have a limit of 40 characters.
- Names in the Application Object Tree (AOT) and in X++ code must correspond to the names in the U.S. English user interface.
- Names must be spelled correctly.
- Names must be used consistently.
- Each path in the AOT must be unique; nodes must not have identical names.
- All texts that appear in the user interface must be defined by using a label. (This rule applies only if you want your solution certified as an international solution.)
- Do not begin a name with "aaa", or "CopyOf". Do not begin a name with "DEL_" unless it is a table, extended data type or enum, and it is needed for data upgrade purposes. Names that use these prefixes cause a best practices error when you check the objects into the version control system.

Rules are available about the following:

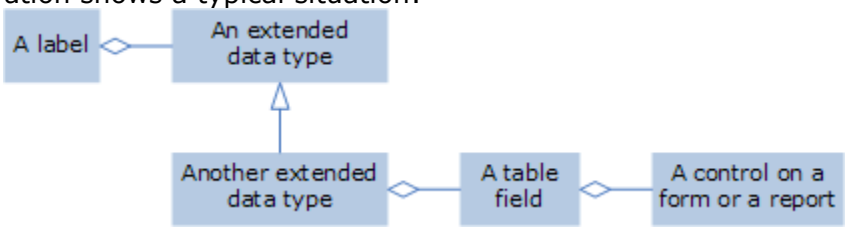
- Use of Labels
- Name structure
- Upper and lower case
- Underscores
- Abbreviations
- Prefixes
- Automatically Generated Names

Best Practices for Labels

Every user interface text must be defined by using a label.
A new label must be created for each new semantic use.
Existing labels must be reused. Always try to find a label that expresses the same semantics as the semantics you want to express.
A label should have an uppercase first letter and all the other letters should be in lowercase.
Labels should not end with a period, unless they are HelpTexts, or unless they end with three periods, for example: "New…", "Add…".
You should not use text constants (for example "%1 - %2") to format labels.
Describe the use of the text that is used in the label in US-English in the **Description** field on the **Label editor** form. This is particularly important for shorter labels where the context is not completely clear. The comment is used when the label is translated to other languages.

Inherit Labels from Underlying Data Types

The contents of label-type properties are usually automatically taken from the underlying definitions. For example, the label used for a field HelpText is often inherited from the HelpText of the underlying extended data type. It is an error to insert the same label as that used in the underlying definition (inherit it, do not duplicate it).
However, the label should be changed, if a different description is needed for the item. The following illustration shows a typical situation:



- A label is defined on an extended data type.
- The extended data type is extended by another extended data type, but as long as the label is not changed, it is inherited and reused.
- That other extended data type is used on a field, and as long as the label is not changed on the field, it is reused again.
- The field is shown using a control on a form, but as long as the label is not changed on the control, it is reused again.
  When possible, a label should be created on an extended data type rather than on a field. If an existing extended data type can be reused, but the existing label is considered unsuitable, create a new extended data type based on the former one and change only the label.

See Also
HelpText Guidelines

HelpText Guidelines

Apply the following guidelines when setting the `HelpText` property for menu buttons, menus, submenus, and menu items in Microsoft Dynamics AX.

- Phrase sentences in the imperative form.
  Example: "Create or update items."
- If the item does not require user interaction, or if the imperative is inappropriate, use a descriptive form.
  Example: "Inventory value for the physically updated quantity (floating value)."

- End sentences with a period or a question mark.
- Avoid using exclamation marks (!), and do not use question marks to shorten a phrase. Example: Avoid "Update?"; use "Use to update." instead.
- When referencing a command or menu item capitalize the name, but avoid any additional accentuation like quotation marks.
- One-word definitions are usually inadequate.

HelpText Examples

Menu and Sub-menu Text Examples

Journals	Journals available for this module.
Inquiries	Inquiries available for this module.
Reports	Reports that can be printed from this module.
Periodic	Jobs available for periodic processing.
Wizards	Wizards available for guiding you through tasks.
Setup	Setup forms with parameters for setting up this module.

Menu Button Text Examples

Post	Select a posting option for the current record.
Setup	Update setup information for the current project.
Inquiries	Get specific information related to the current project.
Functions	Select a function to be applied on the current project.
Print	Select a printing option for the current record.
Transactions	View transactions entered on the current record.

Tip:
If possible, phrase all HelpTexts within a single menu, submenu, or menu button group identically, with a consistent choice of words.

See Also
Best Practices for Labels

Naming Conventions: Name Structure

Where possible, application object names should be constructed hierarchically from three basic components:

- {business area name} + {business area description} + {action performed (for classes) or type of contents (for tables)}
  Examples:
- CustInvoicePrintout
- PriceDiscAdmCopy

- PriceDiscAdmDelete
- PriceDiscAdmSearch
- PriceDiscAdmName
- PriceDiscAdmTrans

Also refer to the general naming conventions.

Use of Uppercase and Lowercase

Application objects names are mixed case. The first letter of the application object is uppercase. The first letter of each internal word is uppercase. For example, AddressFormatHeading, SalesAmount. Application objects include tables, maps, extended data types, base enums, table collections, macros, classes, forms, reports, queries, menus, and menu items.

Methods, system functions, and variables have mixed-case names in with a lowercase first letter. The first letter of each internal word is capitalized. For example, `classDeclaration`, `createProject`.

Primitive types have lowercase names. For example, `str`, `date`, `int`, `real`, `void`, `boolean`. true, false, and null are all lowercase.

Keywords in the X++ language all begin with a lowercase letter. For example, `if`, `while`, `for`, `select`, `ttsCommit`.

Tips:

- Use the function **Add-Ins** > **Source Code Titlecase Update** to "wash" your code to have correct case.
- If you use version control within Microsoft Dynamics AX and the **RunTitleCaseUpdate** option has been set to Yes, errors in the capitalization of the first letter of names are automatically corrected when you check the object in. The **RunTitleCaseUpdate** option is typically set by an administrator. It is available on the **General** tab of the **Version Control Configuration** form, which can be opened from **Tools** > **Development tools** > **Version control** > **Setup** > **System settings**.

Naming Conventions: Underscores

The underscore character ('_') can be used in the following situations.

- When naming formal parameters, it should be used as the first character.
- In the DEL_ prefix.
- Subclasses can have the same name as their parent class, postfixed with a logical name that describes the subclass specialization. The name of the parent class might have to be shortened. In this case, use an underscore between the shortened parent class name, and the rest of the name. For example: `InventUpd_Financial`, `InventUpd_Physical`.
- In an application object that is specialized for a specific country/region, the name is postfixed with underscore and the country/region code. For example: `TaxReport_BE`, `LedgerJournalizeTrans_ES`.

Do not use an underscore in the following situations:

- Beginning of an application object name.
- First character of a variable name in class declarations or methods.
- End of a variable name in class declarations or methods.

See Also

Naming Conventions

Naming Conventions: Abbreviations

Avoid abbreviations unless the abbreviation is much more widely used than the long form, for example: URL or HTML.
If you need an abbreviation, you must comply with the following rules.

- Consistency: if you apply an abbreviation, it should be used everywhere in place of the name. There should be no more than one abbreviation for a name.
- Recognition: the abbreviation should be commonly recognized and understood (at least by Microsoft Dynamics AX users).

A number of abbreviations are standard in the system. These abbreviations must, like other abbreviations, be used consistently and never written in full. Examples:

Full name	Standard abbreviation
Customer	Cust
Payment	Paym
Bill of material	BOM
Number	Num
Warehouse Management System	WMS

See Also
Naming Conventions

Naming conventions: Prefixes

Subject Area Object Prefix

A subject area specific application object is prefixed with the name of the subject area the object belongs to, for example Cust*, Invent*, Ledger*, Proj*, Vend*.
Examples:
WMSOrderSplit
CustBankAccount
CustBalanceCurrency
InventAccountType

Application Area Object Prefix

An application area object is prefixed with the name of the application area the object belongs to, for example Aif*, and Sys*.

The DEL_ Prefix

DEL_ is a special prefix. It is an abbreviation for "Deleted" and is used for application objects that will be deleted in the next version of the product.
DEL_ tables and fields are necessary to allow data update. Such objects allow access to old data that must be migrated to a new location.
When an object with a DEL_ prefix is introduced, the update mechanisms handle changes in the standard application, for example by moving fields and X++ code to the table that replaces the one with the DEL_ prefix. But, if you have written X++ code that references an application object that has been given a DEL_ prefix, you have to update these references yourself.

See Also
Naming Conventions

Automatically Generated Names

All automatically generated names need to be renamed by using the naming conventions for Microsoft Dynamics AX. The following table shows some examples.

From	To
Class1	The logical class name.
method1	The logical method name.
TabPage	The logical tab page name.
Group1	The logical group name in forms.
ReportDesign1	The logical name of the report design. If there is only one report design, you can call it ReportDesign.
Field_1	The logical name of a field on a report design.
Field1	The logical field name in a table.

Naming Conventions for Variables

Variables of general types (primitive and composite data types) should be named logically.
Variables of specialized types (extended data types) should have the same name as the type (which should have a logical name) but starting with a lower case character.
If you have more than one variable of the same specialized type, use logical names that contain the name of the type as a prefix.
Avoid one-character variable names, except for temporary 'looping' variables, like `i` and `j`, or coordinate variables like `x` and `y`.
Examples:

```
InvoiceJour     invoiceJour;
InvoiceLine     invoiceLine;
CustTable       custTable;
CustTrans       custTrans;
MarkupTrans     markupTrans;
int             i;
```

The prefix of the variable can be removed if the name of the variable is still understandable.

See Also
Naming Conventions

Naming Conventions for License Codes

License codes should use the same prefix as the module name.

See Also
Naming Conventions

Designing a Microsoft Dynamics AX Application

This section discusses how to design a Microsoft Dynamics AX application to minimize problems during future updates such as when applying service packs or moving to the next version of the product.

- Modify Objects in the Standard Application
- Modifying User Interface Text
- Design Principles
- Best Practices: Performance Optimizations
- APIs in the Standard Application
- Frameworks Introduction
- Design Guidelines for Cost-Efficient Upgrades

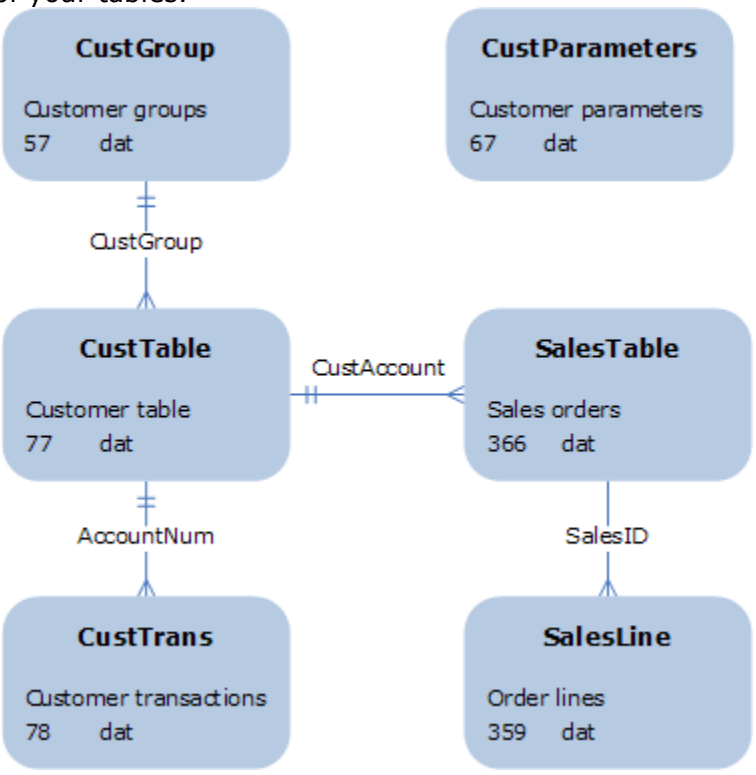
Data Model for New Microsoft Dynamics AX Modules

When creating a new module, create a data model that has the following criteria:

- Has a structure similar to the one in the standard application
- Can be normalized to the third normal form

For a description of the third normal form, see the Microsoft Knowledge Base article about Data Normalization Basics.

The following figure shows a simplification of the basic structure of the data model found in most Microsoft Dynamics AX modules. This structure can be used as a template for setting up the properties of your tables.



Data model found in most modules

The following table shows typical values for the various types of tables used in the previous figure. (All values shown in these tables are only guidelines and might not be exactly as found in the actual tables.)

Property	CustGroup	CustTable	CustTrans
Name postfix	Group.	Table.	Trans.
TableGroup	Group.	Main.	Transaction.
Description	Stores information that categorizes the records.	Stores some base data in the application.	Stores some transactions in the application.
Number of records	Typically relatively low with relatively static information.	Typically high with relatively static information.	Typically very high.
Key	There is always a key.	There is always a key.	There is sometimes a key.
Delete actions	The information is sometimes so non-vital that records can be deleted from the table, even if there are other records in the system that relate to the table.	Restricted against CustTrans and SalesTable.	Not applicable.
CacheLookup	Entire table.	Found, NotInTTS.	None.
ClusterIndex	On the key.	On the key.	Consider.

The following table shows additional typical values for the various types of tables used in the previous figure.

Property	CustParameters	SalesTable	SalesLine
Name postfix	Parameter.	Table.	Trans or Line.
TableGroup	Parameter.	WorkSheetHeader.	WorkSheetLine.
Description	Stores some basic parameters in the application. There is one field per parameter.	Stores some header information for the related worksheet transactions.	Stores worksheet lines in the application.
Number of records	Typically only one, or very few, with very static information.	Typically high with relatively static information.	Typically very high.
Key	There is a key to make the found cache work.	There is always a key.	There is sometimes a key.
Delete actions	Not applicable.	Cascading SalesLine.	Not applicable.
CacheLookup	Found.	NotInTTS.	None.
ClusterIndex	Not applicable.	On the key.	Consider.

See Also
Designing a Microsoft Dynamics AX Application

Modify Objects in the Standard Application

When you modify application objects in the Microsoft Dynamics AX standard application, implement the modifications by using one of the following techniques:

- Modify an existing application object
- Create a new application object that replaces the original application object

If the modification is small, such as adding a few new buttons or some new fields to a form, make the modifications in the original form. This creates a copy of the form in a new layer. If there are many modifications or a total redesign of the form, it is often better to create a new form as a copy of a standard form. You must set the menu item to point to the new object. However, the upgrade wizard does not notify you when the original form changes, which is a disadvantage.

When adding new functionality to an application object, a general strategy is to add a new method, and then call it from the application object. This helps minimize the task of porting the modifications when an upgrade is performed. If the new functionality was implemented directly in one of the existing methods, it would be more time-consuming to port the functionality.

Add Comments

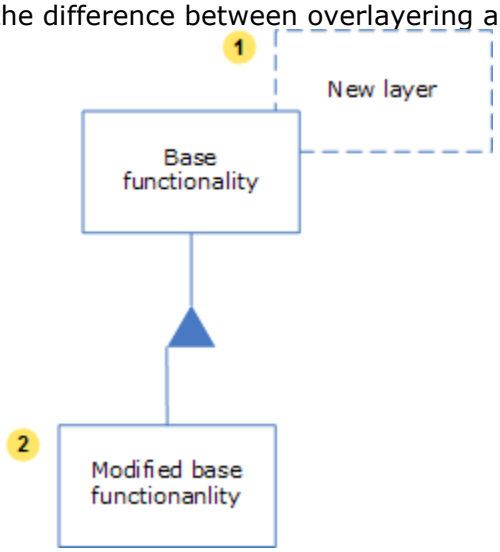
Add a comment whenever a modification is made to code in an application object in the standard application as shown in the following example.

```
void methodName()
{
   // Standard X++ code.

   // <Your Module Name> Begin.
   this.myNewMethod();
   // <Your Module Name> End.

   // Standard X++ code.
}
```

Both the beginning and the end of the new code are marked by the comments. The comments include the name of your custom module.

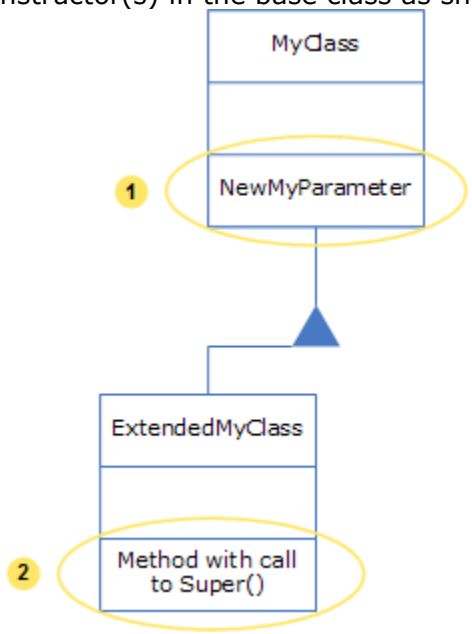
Overlayering and Overriding Classes

The following figure shows the difference between overlayering and extending.



Difference between overlayering and extending

The following information corresponds to the numbers in the previous figure:
1 – Create a new layer if all modules are to use the modified functionality.
2 – Create a subclass to be used instead of the base class only if your new module needs the modified functionality.
If the base class has been constructed by using the ClassFactory Class class, it can have more than one explicit constructor. Instead of creating a new subclass, extend the class and overlayer only the explicit constructor(s) in the base class as shown in the following figure.



Extending a class created by using the ClassFactory class
The following information corresponds to the numbers in the previous figure:
1 - The explicit constructor in the base class is modified (overlayered) to create an object of the `ExtendedMyClass` type.
2 - The methods in the extended class can refer to the base functionality by using `super()`.
By using this technique, overlayer only one method (the explicit constructor in the base class). While you gain all the usual benefits of inheritance, you don't need to modify all the objects that refer to the base class.

See Also
Designing a Microsoft Dynamics AX Application

Modifying User Interface Text

The recommended way to modify text that appears in the user interface is to modify property values on the extended data types or base enums in the application.
When customizing the text, you can create new labels. However, to ensure consistent terminology in the user interface, you should always reuse standard labels, if possible. Never create duplicates.
The strategy is that SYS labels in Microsoft Dynamics AX are never deleted. This means that you can assume that the SYS labels that you use in your application will be available even after a Microsoft Dynamics AX installation is upgraded.
If a customer requests major terminology changes, the fastest solution may be to make the modifications directly in the label file. However, this strategy is not recommended, because all the modifications will be lost when a new label file is installed; for example, when you install a service pack. Microsoft Dynamics AX does not provide any tools that can assist you in maintaining the modifications that you have made manually in the label file.

See Also
Best Practices for Labels

Design Principles

This section describes several design principles to be employed when you design a Microsoft Dynamics AX application.

- Keep Business and User Interface Logic Separate
- Where to Place the Code
- Always Use Field Groups in Tables
- Use Auto Property Settings

See Also
Designing a Microsoft Dynamics AX Application
Frameworks Introduction

Keep Business and User Interface Logic Separate

It is very important to have a clear interface between the presentation logic (user interface) and the business logic. Do not mix these two types of logic.
Business logic must be implemented in classes and on tables.
Never design your business logic so that it has direct references to controls on forms or reports. The design of the business logic must enable any relevant form or report to use it. Remember that the business logic in Microsoft Dynamics AX can be used though COM from other applications. For example, an X++ script can access the business logic that creates a sales quotation. This underscores the importance of not letting the business logic be dependent on a specific Microsoft Dynamics AX form or report.

See Also
Where to Place the Code
Designing a Microsoft Dynamics AX Application

Where to Place the Code

Correct code placement is essential for good application performance (especially in the client/server environment) and to make the application easy to customize, reuse, navigate, and maintain.

3-tier Design

Optimize your programs to utilize the 3-tier architecture that is supported by Microsoft Dynamics AX.

Tier	Objects and code belonging to the tier
Client	Presentation layer. This is where forms are stored. Place client-specific classes and methods here.
Object server	The location of business application logic. Transaction-oriented database update jobs should be placed to run here, close to the database.
Database server	Utilize the power of the database server by using aggregate functions, joins, and other calculating features of the database management system.

Classes

Put code in the classes that are:

- Related to many tables, such as update jobs
  –or–
- Not related to tables, for example, various supporting activities

Create class instance methods:

- When working on the instance variables of the class (objects of)
  –or–
- When overriding is potentially useful

Create class static methods when:

- Access to the class instance members is not needed
- Overriding is not necessary
- The functionality of the method is related to the class it is defined on
- The method might be able to execute on a different tier than the method's

Tables

Put code that is strictly related to a table as methods on that table.
Create table instance methods for handling one record at a time. Create table static methods when handling none, some, or all records.
Split code to be executed on separate tiers into separate methods.

Notes:

- Do not create instance methods when no instance is needed.
- Code in static methods can technically be created anywhere—on any table or class—because it has no physical binding to the instance. Create it either on the table or on the class where it logically belongs.

The Global Class

Place methods in the `Global` class if they cannot be placed more logically on another class (or table).
Methods on `Global` should have the same character as the Functions. They should be general-purpose, tool-extending, and application-neutral.
Do not use the `Global::` prefix when calling `Global` methods—methods on this class do not need a variable declaration.

Forms and Reports

Do not put any code in forms or reports except for calls to the classes and table methods that handle complex layout and business logic.
Do not to place `edit` and `display` methods on forms and reports if they can be placed on a table.
If code cannot be avoided in the form:

- Place the code at the data source/data source field level and not at the control level.
- Call classes from buttons on forms by using menu items (for example, by not using code).

Maps

Use maps for a limited number of connected fields. For example, for the Address fields, code should be placed on `AddressMap`.

Views

Views are limited `select` statements used on read-only tables. Do not place much code on views, unless, for example, you have a `display` method on the parent table.

See Also

Designing a Microsoft Dynamics AX Application

Always Use Field Groups in Tables

When designing tables, it is very important to use field groups. You should also add fields to a field group if you are adding fields to existing tables. Add them to an existing field group, if possible.

When you use field groups, IntelliMorph can automatically update the layout of all related forms and reports whenever a modification is made to the field groups.

If you create new forms or reports based on the field groups defined in the standard tables, you will not have to manually update the custom object when the standard tables are modified by an upgrade.

The sequence of the fields in the field groups on tables can be used to determine the sequence of the fields in forms. To do this, use the `AutoDataGroup` property on the Groups controls in the form design.

This means that the order in which the fields appear in the field group is significant.

See Also

Best Practices for Field Groups

Using Global Variables

Global variables are often needed because of flawed implementation designs. However, if used for caching purposes, global variables can provide increases in performance. This topic describes how you can implement a global variable with zero maintenance during an upgrade.

How to Set the Variable

Get the `globalCache` variable located on the `ClassFactory` class:

```
SysGlobalCache globalCache = ClassFactory.globalCache();
```

Call the `set` method:

```
globalCache.set(str owner, anytype key, anytype value);
```

Parameters	Description
owner	A unique name that identifies you as the user. Use `classIdGet(this)` or `classStr(myClass)`.
key	Identifies your variable. This is useful if you need more than one global variable from the same location.
value	The actual value of your variable.

Get the Variable

Get the `globalCache` variable, located on the `ClassFactory` class:

```
SysGlobalCache globalCache = ClassFactory.globalCache();
```

Call the `get` method:

```
value = globalCache.get(str owner, anytype key, anytype returnValue = '');
```

Parameters	Description
owner	Must be a unique name that identifies you as the user.
key	Identifies your variable.
returnValue	The value you want if the global variable has not been set. This is useful for caching purposes. See the following example.

Example

```
void new(Integer _width  = Imagelist::smallIconWidth(),
         Integer _height = Imagelist::smallIconHeight())
{
    SysGlobalCache globalCache;
    Container      packedData;
    ClassName      className;
    ;

    if (this.keepInMemory())
    {
        globalCache = ClassFactory.globalCache();
        className   = classId2Name(ClassIdGet(this));
        packedData  = globalCache.get(className, 0, connull());
        imageList  = globalCache.get(className+classStr(imagelist),
                                     0,
                                     null);
    }
    if (!imageList)
    {
        imagelist = new Imagelist(_width,_height);
        this.build();
        if (this.keepInMemory())
        {
            globalCache.set(className, 0, this.pack());
            globalCache.set(className+classStr(imagelist),
                            0,
                            imagelist);
        }
    }
    else
    {
        this.unpack(packedData);
    }
}
```

Client/Server Considerations

Because of the duality of `ClassFactory`, global variables can exist either on the client or the server. This could mean less network traffic because it is possible to have the global variable on both sides—set it twice.

To share your global variable across the network and accept the performance penalty of a client/server call, use the `infolog` variable (`Info` class) or `appl` variable (`Application` class) instead of `ClassFactory`. They reside on the client and on the server, respectively.

Use Auto Property Settings

A general design rule is to keep as many property settings set to **Auto** as possible. The benefit of doing this is that the application objects will automatically adjust to any changes of the kernel's interpretation the application object's behavior, such as how a form or report should be displayed.

Use Auto Report Design

When you create reports, it is recommended that you use the Auto design. The benefit of using the Auto design is that your reports will automatically adjust the layout based on the IntelliMorph technology.
For more information, see report design.

Best Practices: Performance Optimizations

This section has some hints on how an application can be optimized for better performance. This topic also contains a list of the best practice checks for performance. For more information, see the following topics:

- Database design and operations
- AOS tuning
- General programming

Design for Performance

Design your application for performance and functionality. Use a good data model, and use the paradigms of MorphX (copy what has already been done). The features of MorphX are built for optimized performance.
Keep in mind the following issues when you create your design.
Network setup:

- The network that connects the client to Application Object Server (AOS) is slow. It is more efficient to make a small number of calls that transfer a large amount of data than it is to make a large number of calls that transfer a small amount of data.
- The network that connects AOS to a database server is fast, but it is quicker to keep calls in the AOS than to call over the network.
- Database servers are usually high-end and fast, but a single database server serves everyone, and represents the traditional performance bottleneck in the application.

Execution of X++ code:

- All X++ statements are fast, but not as fast as compiled and machine-executable C++ code.
- Creation of objects is more time-consuming, but still fast.
- Database-related statements depend on the database design and load. Usually, selects are faster than inserts, inserts are faster than deletes, and deletes are faster than updates.
- Calls between tiers (client/server) are slow.
- Method calls are expensive. Try to reduce the number of calls. For example, do not call the same method several times if you can store a value from the method and instead use that value.

When you design, implement, and test for performance, use a database with a realistic number of records (millions) in the various tables in the database, a realistic number of concurrent users (hundreds), and a realistic configuration of network, clients, and servers.

Best Practice Checks

Microsoft Dynamics AX conducts specific best practice checks for performance. For information about how to set the options for best practice checks, see Best Practice Options. The following table lists the best practice error and warning messages and how to fix the errors.

Message	Message type	How to fix the error or warning
Table is missing Clustered Index	Warning	Using clustered indexes to organize your tables can lead to large gains in performance, but do so carefully. For more information, see Clustered Indexes.
Table is missing Primary Index	Warning	There are advantages and disadvantages for using indexes. For more information, see Best Practices for Indexes.
Extended Data Type is set to be right justified, should be set to left justified	Warning	Set the extended data type to be left-justified. For more information about extended data type best practices, see Extended Data Types Best Practice Checks.
RunBase classes should be able to run on 'Called From' (Ensure pack and unpack are implemented correctly to allow promptPrim to marshal the class across tiers)	Warning	The `RunBase` class is a framework for classes that need a dialog for user interaction and that need the dialog values to be saved per user. For more information, see RunBase Class.

See Also

Designing a Microsoft Dynamics AX Application
Best Practice Compiler Enforced Checks
Best Practices for Microsoft Dynamics AX Development

Performance Optimizations: AOS Tuning

"AOS tuning" is about minimizing the number of calls between the client and the server. The amount of data transported per call is not as important as the number of calls.

Achieve the "select Discount"

Whenever a `select` statement is executed against a data source that is located on another tier, the number of records returned has a certain minimum size (it may return more records than requested). If more records are requested, a new call is performed, and another batch of records are returned for further processing. This effect is referred to as the "select discount", because a round trip to the server is not made for every record fetched, but only for every batch of records.
The "select discount" can be compared to how an insert or an update always cost one call per record.

If you know you only need one record, specify that by using the `firstOnlyselect` statement qualifier.
Normal tables have the data source located on the server. Temporary tables have their data source located at the tier where the code was running at the first call to insert.

Use Containers to Reduce Client/Server Calls

Sometimes you want to transport the values of a large number of variables, or the result of lots of method calls from one tier to another. This costs a lot of client/server calls. The solution can be to use a container holding all the values, transmitted in one call.
Many classes have pack/unpack methods that can be used in such operations.

See Also

Best Practices: Performance Optimizations
Performance Optimizations: General Programming
Performance Optimizations: Database Design and Operations

Performance Optimizations: Database Design and Operations

This topic describes some design techniques that can help to improve database performance.

Caching

Set the table cache level to cache as many records as possible for the table.
Use the record view cache when the same set of records is going to be repeatedly selected.
You can also use your own local caching, in a simple buffer variable, in a record-sorted list or in a temporary table.

Index Design

Index design is very important. Correct index definitions are crucial to a well-performing application. Ensure that there are adequate indexes and that there is the correct number of fields in each index.
It can be useful to add or remove some indexes on the individual installations, depending on the amount and contents of their records.
Using clustered indexes to organize your tables can lead to large gains in performance, but do so carefully. For more information, see Clustered Indexes.

Select Statements

- Use `joins` instead of multiple nested `while` selects.
- Use field lists where applicable.
- Use select/aggregate functions where applicable.
- Use `delete_from` (instead of `while select ... .delete()`).
- Select from the cache where possible.

Transactions

- Make transactions as short and small as possible, to avoid deadlocks and large rollback logs.
- Never wait for a user interaction inside a transaction.
- If several operations must be performed before data returns to a consistent state, perform all the operations in one transaction.
- Avoid deadlocks. Explicitly select records to be updated in a specific order within the table (preferably the order of the key), each time, throughout the application. Select

records for update from different tables in the same order, each time, throughout the application.

Avoid Lengthy Locks on Commonly Used Records

- Use Optimistic Concurrency Control (OCC).
- Update locks on central, commonly used (updated) records that represent bottlenecks in the application.
- Try to avoid updating commonly used records inside transactions.
- Structure the design so that you can use inserts instead of updates. Inserts do not lock objects.
- Place the `select` statement for updates as close to the `ttsCommit` statement as possible, to reduce the amount of time records are locked.

Do Not Write/Update if the Record Has Not Been Changed

The system routinely optimizes database updates by updating a record only if it has actually been changed. If you can do that optimization yourself, it can spare some database operations.

Use Joins in Forms

Use joins in forms, instead of using display methods that contain `selects`.
Using display methods with selects on grids on forms can be slow on thin AOS clients, especially if they are not cached. A call has to be made from the client to the Application Object Server (AOS) and from there to the database server. These calls are multiplied by the number of visible lines in the grid.
If possible, rewrite the form to use a `join` statement between the data sources. There will be only one call to the database when approximately 20 rows are shown in the grid. It is also possible to filter, find, and sort on the joined fields.

Use the Form's Internal Cache

Whenever records are selected from the database and shown in a form, they are cached internally in the form. You can access and use the cached information rather than re-selecting the records.

Example

A form shows data from up to four tables that are related to the table on the first tab page, each on a grid on a separate tab page. The related tab pages are hidden if there is no data to show.
Tab pages can be hidden by selecting a record in the database with the same range as that shown on the tab page. If no records are retrieved, the tab page is hidden `(.visible(false))`. This results in two selects for the same data per tab page.
The hiding of the tab pages can be optimized to only one select per tab page by checking if there is any data in the related internal cache, and then hiding the tab page if there is no data. The actual check is performed by asking if the buffer related to the data source on the tab page is true (with some records to show) or false.

See Also

Best Practices: Performance Optimizations
Performance Optimizations: General Programming
Performance Optimizations: AOS Tuning

Performance Optimizations: General Programming

This topic describes programming techniques that can help to improve performance, and how to inform the user if there is a lengthy operation.

Use Local Caching

You can improve performance by taking a value that is constantly calculated inside a loop out of that loop, and placing it in a variable that is then used inside the loop.
For example, the following method was called from a body line in a report, and used many times when the report was run. The `isoCurrencyCode` method is called on an `Infolog` object, which is bound to the client. So when this report is run on the server, there are two client calls per line.

```
display CurrencyCode currencyCode()
{
    return infolog.isoCurrencyCode()
        ? infolog.isoCurrencyCode()
        : CompanyInfo::find().currencyCode;
}
```

The previous code was changed so that a `currencyCode` variable was declared in the class declaration, initialized (once) in the `init` method, and then used in the `currencyCode` method. This is the code used in the `init` method.

```
public void init()
{
    super();
    ...
    currencyCode = infolog.isoCurrencyCode()
        ? infolog.isoCurrencyCode()
        : CompanyInfo::find().currencyCode;
}
```

This is the new `currencyCode` method. There are no longer any calls to the client.

```
display CurrencyCode currencyCode()
{
    return currencyCode;
}
```

Optimize the Addition of Elements to Containers

Use the `+=` construct to add elements to a container. This construct is optimized by the compiler and results in elements being added much faster than for a *container* `=` *container* `+` *newValue*`;` construct. The difference in performance is particularly important for large containers.
The following example shows the two different ways of adding elements to a container.

```
void containerExample
{
    container c1;
    int a,b;
    ;
    // The non-optimized way to add an element.
    c1 = c1 + a;
    // The optimized way to add an element.
    c1 += b;
}
```

Lengthy Operations

If a user is interacting with the UI and the operation takes longer than one second, use one of these indicators:

- Use `xInfo.startLengthyOperation` to set the mouse cursor to idle.
- Use an hourglass to show that the operation is progressing.
- Use a progress bar to indicate progress.

See Also

Best Practices: Performance Optimizations
Performance Optimizations: AOS Tuning

Performance Optimizations: Database Design and Operations

Swapping Arrays to Disk

When you declare an array, one of your options is to specify how many array items are to be held in memory. The remaining array items are swapped to disk.

Note:

Use this option with caution, as it could lead to excessive disk swapping or excessive memory usage.

A dynamic array is sized according to the largest index ever used in the array. For example, if you use an index of 1000 in an array, the size of the array is set to 1000. If you then use an index of 500, the array size remains 1000.

The general rules are:

- If you use all or nearly all entries in an array, set the memory option to a large number or do not set the option at all.
- If you use few, non-consecutive entries in the array, set the memory option to a small number, such as 1.
- If you use record IDs as indexes, set the memory option to 1. Record IDs are typically very large integers. When you use them as indexes, the size of your dynamic arrays grows unacceptably large.

For example:

```
MyTable myTable;
boolean foundRecord[,1];
;
while select myTable
    where myTable ...
{
    foundRecord[myTable.RecId] = true;
    ...
}
```

See Also

X++ Standards: Arrays

Design Guidelines for Cost-Efficient Upgrades

A Microsoft Dynamics AX application is typically upgraded a number of times during its lifetime. Upgrades may be made to the standard application, or to customized parts of the application. If the application is not designed according to some fundamental principles, upgrading can be expensive and very time consuming.

This section of the SDK describes the relative cost of changing different types of application objects, and describes which layers should be modified in different circumstances.

- Relative Upgrade Costs
- Best Practices for Application Layers

Also refer to Designing a Microsoft Dynamics AX Application.

Note:

For information about the upgrade process, refer to the Microsoft Dynamics AX Implementation Guide.

Relative Upgrade Costs

The following table provides an overview of the relative upgrade cost for a change to an existing Application object.

The upgrade cost is not an absolute value. This table is meant to give only an approximation of the work involved.

Application Object Type	Cost if functionality added	Cost if functionality changed
Form	High	High
Report[1]	High	High
Report Template	Low	Low
Query	Medium	Medium
Menu	Medium	Medium
Table collection	Medium	Medium
Table	Low	Medium
Class	Low	Medium
Extended data Type	Medium	Medium
Base Enum	Medium	Medium
Menu Items	Medium	Medium

[1] Modifications to a report can be done by adding a new design. In this case, the upgrade cost is considered Medium.
If new functionality is added to a Microsoft Dynamics AX application as new stand-alone application objects, the cost of upgrading is likely to be much lower than if the functionality was created by modifying existing objects.

See Also
Design Guidelines for Cost-Efficient Upgrades

Best Practices for Application Layers

This topic describes best practices and upgrade procedures for partners and customers who are working with archived modules.

Customer (Installation) Layers

USR
The customer (or the customer's partner) can make customizations in the USR layer that are unique to the customer's installation.
ID: 50001-60000

CUS
The CUS layer contains application functionality that was either brought or archived as a standard package by a customer.
The customer (or the customer's partner) needs to import the package to the CUS layer.
As new versions of the package are developed and distributed, the customer must import the new version in the CUS layer without overwriting the modifications to the package that might have been made in the USR layer.
Modifications in the USR layer based on application objects in the CUS layer must be updated after the upgrade.
Solutions like this must be supplied to the customers as export files, preferably as an exported project.
ID: 40001-50000

Partner Layers

VAR
The VAR layer contains a partner's modifications to the set of solutions contained in the BUS layer, forming the partner's own all-inclusive solution for one or more customers.
When new versions of the archived standard solutions arrive, they must be imported in the BUS layer, and an upgrade of the VAR layer must be performed.

The upgraded VAR layer must then be distributed to the customers with that particular VAR layer configuration, and the customer's applications must be upgraded.
The partner must supply the customer with the axvar.aod file.
The partner must keep a catalog of the customer's VAR/BUS configurations to update them correctly.
ID: 30001-40000

BUS
The BUS layer contains a solution created by a partner for one or more customers.
The layer can also contain standard solutions that a partner has archived from other partners.
The partner must import the archived solutions to produce the BUS layer. They must also supply the customer with the axbus.aod file.
The partner must keep a copy of the individual axbus.aod files to import new versions of the archived solutions into the existing files, maintaining the internal IDs.
Partners must supply BUS layer standard solutions to other partners as export files, preferably as an exported project.
ID: 20001-30000

Standard Layers

SL1, SL2, and SL3
The SL1, SL2, and SL3 layers are managed by distributors and are used for vertical partner solutions.
Solutions are protected by configuration keys and license codes.

Note:
The **LicenseCode** property can only be set or changed on Microsoft Dynamics AX configuration keys in the Microsoft layers. You cannot use a non-Microsoft layer containing a Microsoft Dynamics AX configuration key.
ID: 1-20000

HFX
HFX is the application layer used for on-demand hot fixes. A hot fix is a single code package composed of one or more files used to address a problem in the product. Using this layer, developers can apply a secure import of .xpo content without interfering with the existing layers or the need to perform a full upgrade. When a roll-up or service pack containing these hot fixes is subsequently released and installed on the system, the HFX layer is automatically emptied.
Solutions are protected by configuration keys and license codes.

Note:
The **LicenseCode** property can only be set or changed on Microsoft Dynamics AX configuration keys in the Microsoft layers. You cannot use a non-Microsoft layer containing a Microsoft Dynamics AX configuration key.
ID: 1-20000

GLS
The GLS layer contains functionality developed by Global Solution Partners and is assembled and supplied as the axGLS.aod file.
Solutions are protected by configuration keys and license codes.

Note:
The **LicenseCode** property can only be set or changed on Microsoft Dynamics AX configuration keys in the Microsoft layers. You cannot use a non-Microsoft layer containing a Microsoft Dynamics AX configuration key.

ID: 1-20000
Labels are in the axGLS*.ald file.
Each solution partner has an individual range of label IDs.

SYS
The SYS layer is the standard application. This layer contains functionality developed and supplied as the default axSYS.aod file.
ID: 1-20000
Labels are in the axSYS*.ald file.

Modified Layers

You always log on to Microsoft Dynamics AX on the current layer. You usually log on to the outermost layer in the installation that you are working on. For certain situations, however, logons are to a different layer.
When you create new application objects, they are placed in the current layer. For example, if you log on to the CUS layer, new application objects are created in the CUS layer.
When you edit application objects, they are placed in the highest layer they are represented in, but not lower than the current layer:

- If you log on to the USR layer and modify any existing application object, the modified application objects are created in the USR layer.
- If you log on to the CUS layer and modify an existing application object from the CUS layer, the modified application objects are replaced in the CUS layer.
- If you log on to the CUS layer and modify an existing application object from the USR layer, the modified application objects are replaced in the USR layer.
- If you log on to the CUS layer and modify an existing application object from the GLS layer, the modified application objects are created in the CUS layer.

When you import application objects, they are placed directly in the current layer:

- If you log on to the USR layer and import some application objects, all the imported application objects are placed in the USR layer:
- Existing application objects in the USR layer are lost and replaced with the imported ones.
- Existing application objects in the CUS layer (or any lower) are overridden with the imported ones in the USR layer.
- New application objects are placed in the USR layer.
- If you log on to the CUS layer and import some application objects, all the imported application objects are placed in the CUS layer:
- Existing application objects in the USR layer are unchanged, but are overridden with the imported ones from the CUS layer. (The USR layer has to be updated.)
- Existing application objects in the CUS layer are lost and replaced with the imported ones.
- Existing application objects in the GLS layer (or any layer lower than CUS) are overridden with the imported objects in the CUS layer.
- New application objects are placed in the CUS layer.

APIs in the Standard Application

The internal APIs (Application Programming Interfaces) in the application must be used when interfacing to their part of the application.
The application must provide APIs for accessing and protecting the functionality in the application that might be of use in other modules.
Characteristics of an internal API are as follows:

- It is class-based.

- The user of the API should only know what the API does or is used for. They shouldn't have to know how the API is implemented.

The purpose of the API can be the following:
- To provide a simple interface to some complex structures.
- To provide a well-defined interface to some widely used functionality.
- To protect the underlying database structures from incorrect use.

Internal APIs

The following internal APIs exist in the standard application.

Ledger

- `LedgerVoucher`
- `LedgerCoverage` (Cash-flow forecast)

Customer

- `CustVoucher`

Vendor

- `VendVoucher`

Inventory

- `InventUpdate/InventMovement`
- `BOMReportFinish`
- `BOMRouteCopyJob`

Sales

- `SalesAutoCreate`

Purchase

- `PurchAutoCreate`

Project

- `ProjPost/ProjTrans`

Number Sequences

- `NumberSeq`

System

- `SysDataImport, SysDataExport`
- `BatchInfo`

Ax<table> Classes

Following is a list of all the Ax<table> classes. These classes should be placed under their module and not Ax<table> classes.

- `AxInternalBase`
- `AxAddress`
- `AxCommissionSalesRep`
- `AxCompanyInfo`
- `AxContactPerson`

- AxCustInvoiceJour
- AxCustInvoiceTrans
- AxCustTable
- AxDocuRef
- AxECPustSignup
- AxEmplTable
- AxInventDim
- AxInventItemLocation
- AxInventTable
- AxInventTableModule
- AxMarkupTrans
- AxPriceDiscTable
- AxProjActivity
- AxProjCategory
- AxProjGroup
- AxPurchLine
- AxPurchTable
- AxSalesLine
- AxSalesTable
- AxVendPurchOrderJour
- AxVendPurchOrderTrans
- AxVendTable

Frameworks Introduction

Frameworks are collections of design patterns, interfaces, actual code, and so on. They make up a system that provides some support for you as a programmer. It is best practice to use and exploit the existing frameworks and subsystems rather than create something similar yourself.
This topic contains short descriptions of some of the frameworks, subsystems, and features in Microsoft Dynamics AX.

Address System

Addresses are uniformly handled by the Address subsystem.

BatchJournal Framework

The BatchJournal Framework enables you to run a group of tasks that have been created by using the RunBase framework. (There are individual `tts` transaction controls for each task.)

Consistency Check Framework—Check and Fix

The Consistency Check Framework helps validate the consistency of the data in the production database and helps fix the inconsistencies that it finds.
The framework consists of classes with names ending in "ConsistencyCheck." For example, `InventConsistencyCheck`.

Dimension

Use the How to: Add Dimensions subsystem to work with financial dimensions. Use the InventDim subsystem to work with inventory dimensions.

Document Management System

The document management system is automatically available everywhere in Microsoft Dynamics AX. Text notes and documents of any type and size can be attached to any record in Microsoft Dynamics AX.
Note fields should not be placed on your own tables. The document management system's text notes are used for that.
Note If documents attached to a record are stored on a file share rather than in the database in Microsoft Dynamics AX, you must ensure that the correct access levels have been set.

Infolog

Information and messages to the user are placed in the Infolog. It also contains contextual information and supporting Help and actions.

Number Sequence

Create a number sequence for a new module by using the Number Sequence Framework.

OLAP

Your application must support OLAP possibilities. To do this, use the (BAS* classes).

Operation progress

Inform the user that a process will take some time by using the How to: Create Progress Indicators (SysOperationProgress).

ReleaseUpdate

The data model in Microsoft Dynamics AX has changed from that in Microsoft Axapta 3 0. Use the ReleaseUpdate framework to update the data in your customers' database.

RunBase

Use the RunBase Framework to execute tasks that cannot be executed in batch.
The framework provides you with dialogs, users last values, query dialogs, and progress bars. It also provides a unified user and programmer experience.
It is best practice to make it possible to execute the jobs in Batch.

RunbaseBatch

Use RunBaseBatch to execute tasks and reports in batches. This enables the user to work with the next task in Microsoft Dynamics AX instead of waiting for another task to finish. All tasks that cannot be executed in batch should be implemented with the RunBase Framework.

RunbaseReportStd

Use the RunBaseReportStd framework to implement all reports.

RunBaseMultiParm

The RunbaseMultiParm framework does the following:

- Enables you to run one or many updates in one job
- Provides a job history
- Provides more straightforward parameter handling

The frameworks use parameter tables that are specifically designed for a particular task. For example, ProdParmBOMCalc, which is used with the `ProdMultiBOMCalc` class.

Instead of packing many parameters, only a parmId is packed. It identifies one or many records in the parameter table.
The parameters needed are created as fields in the parameter table. The dialog is a form in the AOT, showing a grid by using the parameter table instead of a traditional dialog form. For each parameter record, the actual `RunbaseMultiParm` class runs a specific `UpdateBase` class. There will typically be an `UpdateBase` class for each `RunbaseMultiParm` class.

TransactionLog Subsystem

The TransactionLog subsystem is used to offload the transaction details, such as CreatedDate and CreatedBy, from the transaction tables and place them in a central transaction header table. All transactions across the application share a common entry in the transaction header table, and they are related to it.

Web Applications (WebApp)

Create applications for the Web by using the Web Application Framework for Microsoft Dynamics AX. For more information, see Enterprise Portal for Microsoft Dynamics AX.

Wizard

Create wizards by using the Wizard Wizard, and the Wizard framework.

See Also
Microsoft Dynamics AX Class Design Patterns

Number Sequence Framework

This topic describes how to implement the number sequence framework for a new module in Microsoft Dynamics AX. The topic will show how some number sequences could be implemented for the Fleet Management (*FM*) sample module. (Some of the following steps might be irrelevant if they have been previously performed for other purposes in your new Microsoft Dynamics AX module.)

Creating a Parameter Table and a Number Sequence Reference Class

The first step to implementing number sequences for a new module is to create a parameter table and a number sequence reference class.

1. Create a parameter table for the new module: *MyModule*Parameters. For the Fleet Management module, this table is named FMParameters. The table must, at a minimum, contain a Key field and a `find` method. The `delete` and `update` methods must be overridden. These methods can be copied from one of the other parameter tables, such as BOMParameters.

2. Create an enumerated value that represents all of the number sequences for the module by adding a new element to the `NumberSeqModule` base enum. For the Fleet Management module, the *FM* element was added to the base enum.

Note:
Configuration keys are used to detect the active number sequence references in your Microsoft Dynamics AX installation. If the configuration key is not enabled, the modules number sequence references are not displayed in the general References form. The user cannot see references from modules that are not enabled.

3. Create a new number sequence reference class named `NumberSeqReference_`*MyModule*. This class must extend the `NumberSeqReference` class. For the Fleet Management module, this class is named `NumberSeqReference_FM`.

4. Add the `numberSeqModule` method to the new number sequence reference class. This method must return the element for your module from the `NumberSeqModule` base enum. The following code shows how this is done for the Fleet Management module.

```
public static client server NumberSeqModule numberSeqModule()
{
   return NumberSeqModule::FM;
}
```

5. Implement the `numberSeqModule` and `numberSeqReference` methods for the parameters table.

Copy these methods from one of the other parameter tables such as BOMParameters, and then change the names that are found in the method. Change the return value of the `numberSeqModule` method so that it references the number sequence class for your module. For example, the `numberSeqModule` method from the FMParameters table returns `NumberSeqReference_FM::numberSeqModule`. The `numberSeqModule` method of the `NumberSeqReference_FM` class returns `NumberSeqModule::FM` (the FM element of NumberSeqModule).

6. Add *MyModule*`Parameters::find()` as a new line in the `selectParameters` method of the `Company` class. The following example shows the line that was added for the Fleet Management module.

```
FMParameters::find();
```

7. Create a form to display the new parameter table.

It is important that the functionality of number sequence references is copied exactly from one of the other parameter forms (for example, CustParameters). Remember to change the names of the called methods.

8. In the `NumberSeqReference` class, add a new line to the `construct` method—copy one of the existing lines, and then change the name of the class. The following example shows the line that was added for the Fleet Management module.

```
case(NumberSeqReference_FM::numberSeqModule()) : return new NumberSeqReference_FM(_module);
```

9. In the `NumberSeqReference` class, add a new line to the `moduleList` method—copy one of the existing lines, and then change the name to reference your number sequence class. The following example shows the line that was added for the Fleet Management module.

```
moduleList += NumberSeqReference_FM::numberSeqModule();
```

The new number sequence framework is now established. The **Number sequences** tab should display the "No setup required" message.

Making Number Sequence References

Next, you will make number sequence references for the number sequences you are creating for your new module.

1. In your number sequence reference class, override the `loadModule` method.

2. In this new method, specify the characteristics of each number sequence reference you need in the new module. For example, the following code is from the `loadModule` method of the `NumberSeqReference_FM` class. It defines two number sequences used by the Fleet Management module.

```
protected void loadModule()
{
    NumberSequenceReference numRef;
    ;

    // Setup VehicleNum ID
    numRef.DataTypeId               = typeId2ExtendedTypeId(typeid(VehicleNum));
    numRef.ReferenceHelp            = "Unique key for Fleet Management vehicles";
    numRef.WizardContinuous         = false;
    numRef.WizardManual             = NoYes::Yes;
    numRef.WizardAllowChangeDown    = NoYes::No;
    numRef.WizardAllowChangeUp      = NoYes::No;
    numRef.SortField                = 1;

    this.create(numRef);
```

```
    // Setup TripNum ID
    numRef.DataTypeId               = typeId2ExtendedTypeId(typeid(TripNum));
    numRef.ReferenceHelp            = "Unique key for trips";
    numRef.WizardContinuous         = false;
    numRef.WizardManual             = NoYes::Yes;
    numRef.WizardAllowChangeDown    = NoYes::No;
    numRef.WizardAllowChangeUp      = NoYes::No;
    numRef.SortField                = 2;

    this.create(numRef);
}
```

Tip:
For details about how to create a reference, see the comments written above the code for the `NumberSeqReference.loadModule` method.
After creating your references, the system automatically detects them when opening the parameter form. They should now be visible in the grid on the **Number sequences** tab.

Accessing Your New References

For each reference specified in `NumberSeqReference`*MyModule*`.loadModule`, you must create a static method on your parameter table. Assuming that you have specified a reference for the *MyDataType* data type, create the *MyModule*`Parameters::numRef`*MyDataType* method.

1. Copy a `numRef` method from one of the other parameter tables.
2. Change the name of the method to `numRef`*MyDataType*.
3. Add code that will return a number sequence reference object for that specific data type. For example, the following method retrieves the number sequence reference object that is used for the `TripNum` field.

```
server static NumberSequenceReference numRefTripNum()
{
    return NumberSeqReference::findReference(typeId2ExtendedTypeId(typeid(TripNum)));
}
```

Using Number Sequences in an Application

To use the number sequence for a form in Microsoft Dynamics AX or in Enterprise Portal, you will typically add code to the data source for the form or data set. You can also retrieve a number sequence value directly in code. For example, the following example retrieves the next available vehicle number from the number sequence used for the VehicleNum field and displays it in the Infolog.

```
Info(NumberSeq::newGetNum(FMParameters::numRefVehicleNum()).num());
```

Forms

To use a number sequence for a form in Microsoft Dynamics AX, follow these steps.

1. In the `classDeclaration` method of the form that will be accessing data, add a variable declaration for the number sequence handler. The following example shows the variable definition for a number sequence handler.

```
public class FormRun extends ObjectRun
{
    NumberSeqFormHandler numberSeqFormHandler;
}
```

2. Add the `NumberSeqFormHandler` method to the form. The code in this method will create an instance of the number sequence form handler and return it. The following example shows the code that returns the number sequence form handler for the Trips form of the Fleet Management sample module.

```
NumberSeqFormHandler numberSeqFormHandler()
{
    if (!numberSeqFormHandler)
    {
        numberSeqFormHandler = NumberSeqFormHandler::newForm(
            FMTrips::numRefFMTrips().NumberSequence,
             element,
             FMTrips_DS,
             fieldnum(FMTrips, TripNum));
    }
    return numberSeqFormHandler;
}
```

3. Add `create`, `delete`, and `write` methods to the data source of the table that contains the field for which the number sequence is being used. The following code examples show these methods that are added to the data source for the FMTrips table to support the number sequence for the TripNum field.

```
public void create(boolean _append = false)
{
    element.numberSeqFormHandler().formMethodDataSourceCreatePre();
    super(_append);
    element.numberSeqFormHandler().formMethodDataSourceCreate();
}

public void delete()
{
    element.numberSeqFormHandler().formMethodDataSourceDelete();
    super();
}

public void write()
{
    super();
    element.numberSeqFormHandler().formMethodDataSourceWrite();
}
```

Enterprise Portal

To use a number sequence for a form in Enterprise Portal, follow these steps.

1. In the `classDeclaration` method of the data set that will be accessing data, add a variable declaration for the number sequence handler. The following example shows the variable definition for a number sequence handler.

```
public class DatSetRun extends ObjectRun
{
    NumberSeqFormHandler numberSeqFormHandler;
}
```

2. Add the `NumberSeqFormHandler` method to the data set. The code in this method will create an instance of the number sequence form handler and return it. The following example shows the code that returns the number sequence form handler for the FMTripAddEdit data set of the Fleet Management sample module.

```
NumberSeqFormHandler numberSeqFormHandler()
{
    if (!numberSeqFormHandler)
    {
        numberSeqFormHandler = NumberSeqFormHandler::newForm(
            FMTrips::numRefFMTrips().NumberSequence,
            element,
            FMTrips_DS,
            fieldnum(FMTrips, TripNum));
    }
    return numberSeqFormHandler;
}
```

3. Add `create`, `delete`, and `write` methods to the data source for the data set that contains the field for which the number sequence is being used. The following code examples show these methods that are added to the data source for the FMTrips table to support the number sequence for the TripNum field.

```
public void create(boolean _append = false)
{
    element.numberSeqFormHandler().formMethodDataSourceCreatePre();
    super(_append);
    element.numberSeqFormHandler().formMethodDataSourceCreate();
}

public void delete()
{
    element.numberSeqFormHandler().formMethodDataSourceDelete();
    super();
}

public void write()
{
    element.numberSeqFormHandler().formMethodDataSourceWrite();
    super();
}
```

RunBase Framework

The `RunBase` framework provides a standardized approach to creating processes and batch jobs in Microsoft Dynamics AX. The framework must be used for every job-style function in the application.

The framework is implemented by the `RunBase` application class and supplies many features, which include the following:

- Query
- Dialog, with persistence of the last values entered by the user
- Validate
- Batch execution for users to schedule jobs. This functionality uses the RunBaseBatch Class class) and the `pack` and `unpack` methods with versioning.
- Progress bar
- Run
- Client/server-optimized

Following are descriptions of some of the most important `RunBase` methods.

new Method

Always call `super()` in the `new` method.
The `new` method must not be too slow for your needs (it is periodically called for administrative purposes).

description Method

You must create a static `description` method that returns a class description—the class's role in the UI.
The `description` method must have the `RunAs` property set to Called or the equivalent behavior. If the class is defined as `client` or `server`, define the method as `client server`.
For example:

```
client server static ClassDescription description()
{
    return "@SYS54106";
}
```

run Method

The `RunBase.run` method is the central method of the class. The job is done in the `run` method.
Skeleton:

```
void run()
{
    // Local declarations.
    try
    {
        this.progressInit
        ttsBegin;
        // Reset the variables that were changed in the transaction.
        ...
        // Do the job.
        while select forUpdate myTrans...
        {
            progress.incCount();
            progress.setText
            ...
            ...
        ttsCommit;
    }
    catch (Exception::Deadlock)
    {
        retry;
    }
}
```

pack and unpack Methods

For information about the `pack` and `unpack` methods, see the pack-unpack design pattern.

RunBaseReport.initQuery Method

Place code that modifies the RunBase query in the RunBaseReport.initQuery Method method. The RunBaseReport.initQueryRun Method method calls `initQuery`.

If the query in a `RunbaseReport` depends on values from the dialog, call `this.initQueryRun` after the data is moved from the fields to the variables.

Example

```
private boolean getFromDialog()
{
    ;
    perDate = dialogDate.value();
    this.initQueryRun();
    return super();
}
```

The `initQueryRun` method in the previous code calls `initQuery`.

```
private Query initQuery()
{
    query query = super();
    queryBuildRange qbr;
    ;
    qbr = query.dataSourceTable(
        tableNum(InventTrans)).findRange(
            fieldNum(InventTrans,DatePhysical));
    if (!qbr)
    {
        qbr = query.dataSourceTable(
            tableNum(InventTrans)).addRange(
                fieldNum(InventTrans,DatePhysical));
    }
    qbr.value(SySQuery::range(prevYr(PerDate),PerDate));
    qbr = query.dataSourceTable(
        tableNum(InventTrans)).findRange(
            fieldNum(InventTrans,DateFinancial));
    if (!qbr)
    {
        qbr = query.dataSourceTable(
            tableNum(InventTrans)).addRange(
                fieldNum(InventTrans,dateFinancial));
    }
    qbr.value(SysQuery::value(perDate+1)
    + '..' + ',' + sysQuery::valueEmptyString());
    return query;
}
```

Best Practices: Application Integration Framework

Microsoft Dynamics AX provides a framework called Application Integration Framework (AIF) to integrate with an external application. You can use AIF to send and receive XML documents that represent business objects, such as a customer or vendor. To process these documents, Microsoft Dynamics AX must be able to parse and generate XML. AIF provides automatically generated classes referred to as data objects to implement XML serialization and de-serialization. Data objects can be defined from multiple artifacts. For more information, see Application Integration Framework Overview.

Microsoft Dynamics AX conducts a best practice check to verify that the data object is synchronized with the underlying artifacts that were used to define the data object. The

best practice check is for missing or extra methods in the data object. For information about how to set the options for best practice checks, see Best Practice Options.

Best Practice Checks

The following table lists the best practice error messages and how to fix the errors.

Message	Message type	How to fix the error or warning
Data object class %1 is missing method %2.	Error	Use the **Update document service** form to synchronize the data objects with the underlying schema.
Data object class %1 has extra method %2.	Error	Use the **Update document service** form to synchronize the data objects with the underlying schema.

Long-Running BP Checks

Best practices checks on certain AIF objects such as `Axd`<*Document*> classes, `Ax`<*Table*> classes, and Axd queries can take a very long time. If best practices checks are enabled as part of compiling, then compiling these classes can also take a long time. In addition, using the AIF Document Service Wizard can take longer than expected because of the best practice checking that it performs.

When you encounter a long-running best practices check, you can:

- Allow the best practices check to finish.
- Do not run the best practices check.
- Do not compile with best practices checks on.
- Disable the best practices rule for data objects in the **Best Practice parameters** form. To open this form, go to **Tools** > **Options** and click **Best Practices**. In the best practices tree, expand the **Application Integration Framework** node, and then clear the **Entity and Data Objects Classes** field.

See Also

Best Practices for Microsoft Dynamics AX Development
Best Practice Options

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989
Worldwide +1-701-281-6500
www.microsoft.com/dynamics

Microsoft