Protocol Development Quick Checklist (with corresponding guidelines)

Strictly confidential

Contents

[1 Introduction 5](#_Toc94857008)

[2 Protocol Development Checklist 6](#_Toc94857009)

[2.1 ID 6](#_Toc94857010)

[2.2 General 6](#_Toc94857011)

[2.3 Protocol 6](#_Toc94857012)

[2.4 Validation 7](#_Toc94857013)

[2.5 Points of Attention (for Skyline Developers only) 8](#_Toc94857014)

[3 Protocol Development Coding Guidelines 9](#_Toc94857015)

[3.1 General 9](#_Toc94857016)

[3.1.1 Protocol Name 9](#_Toc94857017)

[3.1.2 Formatting 9](#_Toc94857018)

[3.1.3 Names 9](#_Toc94857019)

[3.1.4 ID Values 11](#_Toc94857020)

[3.1.5 C# Code Conventions 12](#_Toc94857021)

[3.2 Administrative Metadata 22](#_Toc94857022)

[3.2.1 Comment Metadata 22](#_Toc94857023)

[3.2.2 Protocol Metadata 22](#_Toc94857024)

[3.3 User Interface 22](#_Toc94857025)

[3.3.1 Element Wizard 22](#_Toc94857026)

[3.3.2 Protocol Pages 23](#_Toc94857027)

[3.3.3 Displayed Text 23](#_Toc94857028)

[3.3.4 Parameters 24](#_Toc94857029)

[3.3.5 Tables 25](#_Toc94857030)

[3.3.6 Buttons 26](#_Toc94857031)

[3.3.7 Toggle Buttons 26](#_Toc94857032)

[3.3.8 Page Buttons 26](#_Toc94857033)

[3.3.9 Progress Bars 26](#_Toc94857034)

[3.4 Monitoring 26](#_Toc94857035)

[3.4.1 Trending 26](#_Toc94857036)

[3.4.2 Alarming 27](#_Toc94857037)

[3.5 Protocol 27](#_Toc94857038)

[3.5.1 API 27](#_Toc94857039)

[3.5.2 Communication 27](#_Toc94857040)

[3.5.3 Logic 27](#_Toc94857041)

[3.5.4 Timers 28](#_Toc94857042)

[3.5.5 DVE 28](#_Toc94857043)

[3.5.6 Tables 29](#_Toc94857044)

[3.5.7 Parameters 31](#_Toc94857045)

[3.5.8 QActions 31](#_Toc94857046)

[3.5.9 .NET Recommendations 35](#_Toc94857047)

[3.6 Validation 40](#_Toc94857048)

[3.6.1 General 40](#_Toc94857049)

[3.6.2 Spell Checker 40](#_Toc94857050)

[3.6.3 DIS Validator 40](#_Toc94857051)

[3.6.4 Stream Viewer 40](#_Toc94857052)

[3.6.5 Log Files 40](#_Toc94857053)

[3.6.6 Alarm Console 40](#_Toc94857054)

[3.7 Points of Attention 40](#_Toc94857055)

[3.7.1 Skyline Driver Passport Platform 40](#_Toc94857056)

[3.7.2 Dojo Use Cases 40](#_Toc94857057)

[4 Abbreviations 41](#_Toc94857058)

Revisions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Revision | Date | | Name | | Comments | |
| 001 | 08.10.2018 | | KA/MGO | | Initial version | | |
| 002 | 22.05.2019 | | SVD/JST | | Revision | | |
| 003 | 24.09.2019 | | KA | | Revision | | |
| 004 | 17.02.2020 | | KA/MGO | | Revision + template update | | |
| 005 | 06.05.2020 | | JST/MGO | | Item removed from guidelines for parameter description and values | | |
| 006 | 28.09.2020 | | KA/MGO | | Guidelines updated under Administrative Metadata Element wizard > Connection Names | | |
| 007 | 06.01.2020 | | JST/MGO | | Check SCR1249 removed | | |
| 008 | 02.04.2021 | | PDB/MGO | | Note added on XML documentation | | |
| 009 | 12.05.2021 | | JST/MGO | | SCR1250 and SCR1012 updated | | |
| 010 | 31.05.2021 | | PDB/MGO | | Obsolete DCF guideline removed | | |
| 011 | 06.07.2021 | | JST/MGO | | SCR1013 removed | | |
| 012 | 19.10.2021 | PDB/MGO | | SCR1257 added, administrative metadata section updated, title case section updated, minimum DMA version section removed, Dojo use cases section added | |
| 013 | 10.12.2021 | JST/MGO | | SCR1141: Info added on inter-process performance | |
| 014 | 04.02.2022 | JST/MGO | | Obsolete checks removed | |

# Introduction

This checklist must be used during any DMS Protocol related tasks where quick QA is required.

Each check has the scope “**Protocol**” and needs to be verified against the entire protocol for errors that have a critical impact on the system or may prevent the protocol from working. Such a check may also be related to something that needs to be fixed for administrative reasons.

Some checks may require you to implement an impacting change. Applicable checks contain the following warning:

***Warning: Possible impact on existing platforms if changed in existing protocol.***

You must verify every item in the protocol development checklist and specify “OK”, “Fail” or “NA” every time.

The guidelines specified in this document are designed to uphold the level of quality of DMS protocols by improving readability, maintainability and performance.

By adhering to these guidelines, we can ensure that protocols have a consistent look, and thereby increase maintainability and readability. The guidelines also increase reusability, stability and performance.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (see <https://www.ietf.org/rfc/rfc2119.txt>).

# Protocol Development Checklist

|  |  |
| --- | --- |
| **Note**: | For each of the items in this checklist, a link is provided to the appropriate section in the guidelines. After you have clicked such a link to check the guideline in question, use the key combination *Alt* + *Left arrow* to instantly return to the location in the checklist where you clicked the link. |

## ID

|  |  |
| --- | --- |
| **Protocol name** | SLC SDF DCF |
| **Protocol version** | 1.0.0.1 |
| **SLC integration engineer** | JST |
| **Date** | 04.10.2022 |
| **Remarks** | Converted version from Examples/Protocols/DCF version 1.0.0.8 |

## General

| # | Scope | Description | OK | Fail | NA | Remarks |
| --- | --- | --- | --- | --- | --- | --- |
| **Names** | | | | | | |
| {SCR1029} | Protocol | DMS element names are unique, are not empty and contain no leading/trailing spaces or disallowed characters. [(Ref. 3.1.3.6 DMS Element Names)](#_Parameter_Names)  ***Warning: Possible impact on existing platforms if changed in existing protocol.*** |  |  |  |  |

## Protocol

| # | Scope | Description | OK | Fail | NA | Remarks |
| --- | --- | --- | --- | --- | --- | --- |
| **Logic** |  |  |  |  |  |  |
| {SCR1114} | Protocol | No operation takes longer than the specified Run-Time Error (RTE) duration. [(Ref. 3.5.3.1 Operation duration)](#_Fragmenting_Operations) |  |  |  |  |
| **DVE** | | | | | | |
| {SCR1130} | Protocol | DVE child elements are not deleted automatically by default. [(Ref. 3.5.5.3 DVE Child Element Deletion)](#_DVE_Child_Element) |  |  |  |  |
| **Tables** | | | | | | |
| {SCR1137} | Protocol | The primary keys are unique, do not include prohibited characters and are as compact and efficient as possible. [(Ref. 3.5.6.1 Primary Keys)](#_Primary_Keys)  ***Warning: Possible impact on existing platforms if changed in existing protocol.*** |  |  |  |  |
| {SCR1141} | Protocol | The appropriate table update strategy has been implemented. [(Ref. 3.5.6.4 Data Handling)](#_Data_Handling) |  |  |  |  |
| **QActions** | | | | | | |
| {SCR1194} | Protocol | Multi-threaded implementations are implemented correctly (e.g. no threads outlasting the QAction) [(Ref. 3.5.8.10 Multi-threading)](#_Multi-threading) |  |  |  |  |
| {SCR1195} | Protocol | Critical sections are implemented correctly. [(Ref. 3.5.8.10 Multi-threading)](#_Multi-threading) |  |  |  |  |
| {SCR1196} | Protocol | A time to sleep of at least 15 ms or a multiple of 15 ms is used with Thread.Sleep calls. A valid reason is specified to use Thread.Sleep [(Ref. 3.5.8.11 Thread.Sleep)](#_Thread.Sleep) |  |  |  |  |

## Validation

| # | Scope | Description | OK | Fail | NA | Remarks |
| --- | --- | --- | --- | --- | --- | --- |
| **General** | | | | | | |
| {SCR1090} | Protocol | No problems are indicated in the Alarm Console. [(Ref. 3.6.6 Alarm Console)](#_Alarm_Console) |  |  |  |  |

## Points of Attention (for Skyline Developers only)

| # | Scope | Description | OK | Fail | NA | Remarks |
| --- | --- | --- | --- | --- | --- | --- |
| **Registration** | | | | | | |
| {SCR1002} | Protocol | A driver help page is available on DCP. In case of a DVE protocol, each exported protocol has a page. |  |  |  |  |
| **Simulation** | | | | | | |
| {SCR1012} | Protocol | For protocols containing connections other than HTTP or SNMP, a static simulation is available in the [designated shared folder.](file:///S:\Public\DataMiner%20Software\Simulations\Device%20Simulations) |  |  |  |  |
| **Pitfalls** | | | | | | |
| {SCR1131} | Protocol | When a DVE protocol uses “noElementprefix”, the unique element name is enforced in the DMS through the use of DMS Notify 72 as described in the Protocol Development Guide.  ***Warning: Possible impact on existing platforms if changed in existing protocol.*** |  |  |  |  |

# Protocol Development Coding Guidelines

## General

### Protocol Name

* The protocol name (just like a trend or alarm template name) must not start with “Production”.

### Formatting

#### Indentation

* Indentation in the protocol consists of either only tabs or only sequences of four spaces.

#### Alignment and Wording

* Alignment and wording must be consistent.

### Names

#### Unique and Meaningful Names

* A protocol consists of different items such as parameters, actions, QActions, etc. Every item has a name that is used for internal reference in e.g. SLProtocol or DIS. Protocol item names must be unique per item type and must be meaningful.

#### Parameter Names

* Parameter names must be well-chosen, describing the value that the parameter will hold. Note that this is also important to improve the readability of code in Quick Actions:
  + The Parameter class (Skyline.DataMiner.Scripting) generates constant fields for parameters defined in a protocol. The name of a field corresponds to the name of the parameter, excluding spaces and symbols.
  + The SLProtocolExt class generates properties for parameters defined in a protocol. The name of a property corresponds to the name of the parameter, excluding spaces and symbols.

**Prohibited names**

The following strings must not be used as parameter names or parameter descriptions. Note that the items in the table should be considered case-insensitive (E.g. \_clients\_connected must also not be used).

|  |  |  |
| --- | --- | --- |
| \_\_Clients\_connected | DMA Redundancy Status | Redundancy Group added |
| \_\_Communication\_DataMiner\_RX | DMS Revisioned | Scheduler info |
| \_\_Communication\_DataMiner\_TX | Document added | Scheduled Task Created |
| \_\_Communication\_Device | Document edited | Scheduled Task Updated |
| \_\_Communication\_Device\_DataMiner\_RX | Document removed | Scheduled Task Deleted |
| \_\_Communication\_Device\_Iteration | Edited | Security Edited |
| \_\_Communication\_Device\_RTT | Element alarm state | Service Templates |
| \_\_Communication\_info\_state | Element Connections Edited | Script Added |
| \_\_Communication\_Message\_Drops | Element created | Script Deleted |
| \_\_Communication\_Session\_DataMiner\_TX | Element disconnection | Script Edited |
| \_\_Communication info | Element masked | Script execution failure |
| \_\_Element id | Element unmasked | Script started |
| \_\_Element Latch state | Entered Prioritized Mode | Service added |
| \_\_Element Latch state | Error during synchronization | Service path changed |
| \_\_Element\_Priority | File changed | Set as production protocol |
| \_\_Element\_Priority | Filter added | Set Parameter |
| \_\_Element RCA Level | Filter edited | SMS Received |
| \_\_Increment\_PID | Filter deleted | SMS Sent |
| \_\_Last\_reset\_time\_pi\_alarm | GSM Signal Strength | SNMP-Managers edited |
| \_\_Nbr\_of\_alarms | GSM General Information | SNMPAgent |
| \_\_PID | Import elements | Spectrum Monitor Created |
| \_\_Reset\_alarms | Information.xml assigned | Spectrum Monitor Deleted |
| \_\_Start\_time\_first\_alarm | Information Added | Spectrum Monitor Edited |
| \_\_Start\_time\_last\_alarm | Information Deleted | Spectrum Monitor Failure |
| \_\_Timer\_base | Information Edited | Spectrum Script Added |
| \_\_Properties | IP Settings | Spectrum Script Edited |
| \_\_Property name | Latch reset info | Spectrum Script Deleted |
| \_\_Property type | Left Prioritized Mode | Start Element Failed |
| \_\_Property value | Link file | Start synchronization |
| \_\_Read out properties | Linked to | Startup DataMiner Agent |
| Alarm colors edited | Load Element Failed | Startup error |
| Alarm Template Added | Load Protocol Failed | STATE |
| Alarm Template Assigned | lock\_status | State change |
| Alarm Template Deleted | lock\_status | Stop DataMiner |
| Alarm Template Edited | lock\_owner | Synchronization finished |
| Annotations Edited | Map Configuration | Table Repair |
| Asset Manager Configuration | Mobile gateway | Task started |
| Automation info | Mobile Gateway lost contact with DataMiner | TIMEOUT |
| Backup status | Nbr of alarms | TotalNbrOfActiveAlarms |
| Client disconnected | New client registered | TotalNbrOfActiveCriticalAlarms |
| Client Eventing | New Element connection | TotalNbrOfActiveMajorAlarms |
| Client notification | No connection with DMA | TotalNbrOfActiveMaskedAlarms |
| Collaboration Message | Notification | TotalNbrOfActiveMinorAlarms |
| Connection established with DMA | Parameter descriptions | TotalNbrOfActiveWarningAlarms |
| Correlation engine | Preset Created | Trending Template Added |
| Database | Preset Edited | Trending Template Assigned |
| Database optimization | Preset Renamed | Trending Template Deleted |
| Database stack | Preset Deleted | Trending Template Edited |
| DataMiner Agent found | Protocol Added | User settings |
| DataMiner Agent lost | Protocol Deleted | VDX Deleted |
| DataMiner run-time | Protocol Edited | VDX Added |
| Database settings edited | Protocol Replaced | VDX Edited |
| Deleted | Real-time TCP Socket | Views Edited |

#### QAction and Action Names

* A meaningful name must be chosen for QActions or Actions, indicating the functionality that is executed. The name of an action should contain a verb (e.g. CheckMaximumDuration, InitiateImageTransfer, etc.).

#### Trigger Names

* The name of a trigger must reflect what it is triggering on (e.g. On Poll Counter Change).

#### Group Names

* The name of a group must clearly describe its content.

#### DMS Element Names

* Names of DataMiner elements must be unique and the following restrictions apply when an element is created by a protocol QAction or as a DVE.

1. No leading and/or trailing spaces.
2. No empty names.
3. Disallowed characters: [\](file:// ); /  :  \*  ?  "  <  >  | ; °
4. No duplicate names: an element must be unique in the DMS.

### ID Values

#### ID Range

* A protocol consists of different items such as parameters, actions, QActions, and so on, where every item has a unique ID. IDs must be within the following ranges:

|  |  |  |  |
| --- | --- | --- | --- |
| Range | Owner | Minimum DMA version | Allowed |
| [1–63999] | Protocol | 1.0.0 | Yes |
| [1 000 000–9 999 999] | Protocol | 9.0.4 | Yes |

The following table gives an overview of the available parameter IDs that can be used by a protocol.

|  |  |  |  |
| --- | --- | --- | --- |
| Range | Owner | Minimum DMA version | Allowed |
| [1–63999] | Protocol | 1.0.0 | Yes\* |
| [64000–64299] | DataMiner/Protocol | 1.0.0 | Yes\*\* |
| [64300–69999] | DataMiner | 1.0.0 | No |
| [70000–79999] | Protocol | 7.5.6 | Yes\*\*\* |
| [80000–99999] | DataMiner/Protocol | 9.0.4 | Yes\*\*\*\* |
| [100000–999999] | DataMiner | 9.0.4 | No |
| [1 000 000–9 999 999] | Protocol | 9.0.4 | Yes\*\*\*\*\* |

\* In general, the range [1–63999] can be used in a protocol for parameters. However, for some specific types of protocols, additional restrictions apply:

|  |  |  |  |
| --- | --- | --- | --- |
| Protocol type | Restricted Range | Owner | Description |
| Enhanced Service | [1–999] | DataMiner | This range must only be used to communicate with DataMiner via predefined parameters. |
| Spectrum Analyzer | [50000–60000] | DataMiner | This range is used by the SLSpectrum process to create dynamic parameters for spectrum monitoring results. |
| SLA | [1–2999] | DataMiner | Parameters in this range are used for communication between DataMiner and SLA drivers. |
| Aggregation | [1–4999] | DataMiner | Parameters in this range are used for communication between DataMiner and aggregation drivers. |

\*\* Only to be used for communication with DataMiner modules. This range contains parameters that can be implemented in protocols to communicate with DataMiner (e.g. a spectrum analyzer). A registry of assigned parameters is maintained. Only these specific parameter IDs can be implemented in the protocol.

\*\*\* Only to be used in mediation/base protocols.

\*\*\*\* Only to be used for communication with DataMiner modules. This range contains parameters that can be implemented in protocols to communicate with DataMiner (e.g. enhanced service drivers, spectrum analyzer drivers, ticketing drivers, etc.). A registry of assigned parameters is maintained. Only these specific parameter IDs can be implemented in the protocol.

\*\*\*\*\* Must only be used in case all parameters in the range [1–63999] are already used.

* IDs from the range [1 000 000–9 999 999] must only be used in case all IDs from the range [1–63999] are already used.
* Protocol.ParameterGroups: IDs larger than or equal to 10000 must not be used for existing protocols or protocols intended to be used on DataMiner Agents prior to version 9.0.4.

#### ID Gaps

* Large gaps must not occur in ID values of related parameters.

#### ID Ordering

* Items must be ordered so that the IDs are monotonically increasing. The only allowed exception is in case of read/write parameters that have a fixed offset.

#### ID Grouping

* Items with the same functionality or purpose should be grouped together.

### C# Code Conventions

Try to use the C# code style guidelines described below. From 2018 onwards, these guidelines must always be followed.

#### General

* Meaningful names should be used, denoting the semantics of the identifier.
* Readability should be favored over brevity.
* Use of underscores, hyphens and other non-alphanumeric characters should be avoided.
* Use of abbreviations or contractions as part of variable names should be avoided (e.g. use “counter“ instead of “cntr”).
* Only acronyms and initialisms that are generally accepted in the field of computing, broadcasting, etc. may be used.

#### Capitalization style

Throughout this chapter, two capitalization styles are mentioned: Pascal case and Camel case.

* Pascal case (hereafter referred to as PascalCasing) refers to the capitalization style where the first letter of the identifier and the first letter of each subsequent concatenated word are capitalized.
* Camel case (hereafter referred to as camelCasing) refers to the capitalization style where the first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized.

|  |  |
| --- | --- |
| Casing | Identifier type |
| PascalCasing [(SA1300)](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1300) | Namespace, Type (Class, Interface, Struct, Enumeration), Method, Property, Event |
| camelCasing | Parameter  Local variable |

***Compound words***

* Words that are part of a closed-form compound word (i.e. a compound word written as a single word, e.g. endpoint) should not be capitalized, but should be treated as a single word.

***Acronyms and Initialisms***

* Acronyms or initialisms consisting of only two characters should be capitalized (e.g. “PollingIP”). PascalCasing or camelCasing should be used for acronyms that consist of more than two characters (e.g. “SnmpResponse” or “snmpResponse” instead of “SNMPResponse”).

#### Namespaces

* Components of a namespace name should use PascalCasing and should be separated with periods. In case a component uses nontraditional casing, this casing should be adhered to (even if it deviates from the above-mentioned casing rules).
* The name of a namespace should be different from the name of types defined in that namespace.
* In case a generic QAction contains code related to different topics (e.g. a general QAction defining some methods to process an XML response and a number of classes defining the data model used in the driver), different namespaces should be introduced so related things can be grouped together. The name of the namespace should start with *Skyline.Protocol*.

#### Classes, Structs and Interfaces

* PascalCasing should be used for names of classes and structs.
* Classes and structs should be named with nouns or noun phrases (e.g. Alarm, Service, TransportStreamParser).
* Interfaces should be named with adjective phrases (or occasionally with nouns or noun phrases) (e.g. ISortable).
* Interface names should be prefixed with the letter “I” ([SA1302](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1302)). This clearly indicates that the type is an interface.
* When a class-interface pair is defined where the class is a standard implementation of the interface, the class and interface name should only differ by the prefix “I”.
* When generic type parameters are used, a single letter “T” should be used as the name in case the meaning is completely clear. In case the meaning is not clear, a more descriptive name should be used and this name should start with a prefix “T”.

#### Enumerations

* Enumeration type names should not use an “Enum”, “Flag” or “Flags” suffix.
* A singular type name for an enumeration should be used unless the values are bit fields.
* A plural type name for flags enumerations should be used (enumeration with bit fields as values).
* Enumeration identifiers should not have an “Enum”, “Flag” or “Flags” suffix.

#### Methods

* PascalCasing must be used for method names.
* Method names should be verbs or verb phrases (e.g. “TransferImage”).

#### Properties

* PascalCasing must be used for property names.
* A property should be named using a noun, noun phrase or adjective.
* Collection properties should be named with a plural phrase instead of a singular phrase followed by the collection type (e.g. “TransportStreams” instead of “TransportStreamList”).
* Boolean properties should start with a prefix “Is”, “Has”, “Can”, “Supports”, “Allows”.

#### Fields

* Constants, non-private read-only fields and static read-only fields must always start with an uppercase letter ([SA1303](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1303), [SA1304](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1304), [SA1311](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1311)).
* Field names must not use Hungarian notation ([SA1305](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1305)).
* Private read-only fields must start with a lowercase letter ([SA1306](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1306)).
* Public or internal fields must always start with an uppercase letter ([SA1307](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1307)).
* Field names should be either a noun, a noun phrase or an adjective.
* Variable names must not be prefixed ([SA1308](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1308)): member variables must not begin with 'm\_' or 's\_' syntax.

#### Parameters

* camelCasing must be used for parameter names.

#### Local Variables

Local variables must have a meaningful name and camelCasing must be applied.

***Hungarian notation***

Names of local variables may start with a prefix denoting their type (so-called Hungarian notation), but this is not required.

Hungarian notation allows related variables to have the same name, except for the type prefix. For example, suppose there is a string variable “sStartTime”, representing a time. The corresponding variable of type DateTime can then be named “dtStartTime” or “startTime”. Without the prefix, it is more difficult to have a unique name for both variables.

In general, using Hungarian notation for all variable names is *not encouraged*. However, when you do use Hungarian notation, you must use the following prefixes:

* ***Built-in Types***

|  |  |  |  |
| --- | --- | --- | --- |
| C# Type | .NET Framework Type | Prefix | Example |
| [bool](https://msdn.microsoft.com/nl-be/library/c8f5xwh7.aspx) | System.Boolean | b | bIsBusy |
| [byte](https://msdn.microsoft.com/nl-be/library/5bdb6693.aspx) | System.Byte | by | byCommand |
| [sbyte](https://msdn.microsoft.com/nl-be/library/d86he86x.aspx) | System.SByte | sby | sbyCommand |
| [char](https://msdn.microsoft.com/nl-be/library/x9h8tsay.aspx) | System.Char | c | cProtocolType |
| [decimal](https://msdn.microsoft.com/nl-be/library/364x0z75.aspx) | System.Decimal | dec | decSignalStrength |
| [double](https://msdn.microsoft.com/nl-be/library/678hzkk9.aspx) | System.Double | d | dSignalStrength |
| [float](https://msdn.microsoft.com/nl-be/library/b1e65aza.aspx) | System.Single | f | fSignalStrength |
| [int](https://msdn.microsoft.com/nl-be/library/5kzh1b5w.aspx) | System.Int32 | i | iRowCount |
| [uint](https://msdn.microsoft.com/nl-be/library/x0sksh43.aspx) | System.UInt32 | ui | uiRowCount |
| [long](https://msdn.microsoft.com/nl-be/library/ctetwysk.aspx) | System.Int64 | l | lErrorCount |
| [ulong](https://msdn.microsoft.com/nl-be/library/t98873t4.aspx) | System.UInt64 | ul | ulErrorCount |
| [object](https://msdn.microsoft.com/nl-be/library/9kkx3h3c.aspx) | System.Object | o | oResponse |
| [short](https://msdn.microsoft.com/nl-be/library/ybs77ex4.aspx) | System.Int16 | sh | shStatusCode |
| [ushort](https://msdn.microsoft.com/nl-be/library/cbf1574z.aspx) | System.UInt16 | ush | ushStatusCode |
| [string](https://msdn.microsoft.com/nl-be/library/362314fe.aspx) | System.String | s | sResponse |

* ***Other***

|  |  |  |
| --- | --- | --- |
| Class | Prefix | Example |
| System.DateTime | dt | dtNow |
| System.TimeSpan | ts | tsHour |
| System.Text.StringBuilder | sb | sbResponse |
| System.Xml.Linq.XDocument  System.Xml.Linq.XElement  System.Xml.Linq.XAttribute | x | xDocument |

* ***Arrays***

For array variables, the type prefix must be preceded by an additional “a” to indicate an array (e.g. string[] asServiceNames).

* ***Strongly Typed Collections (System.Collections.Generic)***

The following table gives an overview of the different prefixes that should be used for collections. In case a collection has only one type (e.g. HashSet<T>), the identifier should include the type prefix (e.g. HashSet<string> hssDevices).

Plural nouns should be used for lists and sets.

|  |  |  |
| --- | --- | --- |
| Class | Prefix | Example |
| Dictionary<TKey, TValue> | dict | Dictionary<string,string> dictMappings |
| HashSet<T> | hs | HashSet<string> hssDeviceTypes |
| LinkedList<T> | ll | LinkedList<string> llsNames |
| List<T> | l | List<string> lsNames |
| Stack<T> | s | Stack<string> ssCalls |
| Queue<T> | q | Queue<string> qsMessages |

* ***Private fields***

Private fields may start with an underscore (“\_”), followed by the prefix defined above if applicable. However, the use of “this.” is favored over the use of underscores to denote a local class field.

* ***Booleans***

The name of a local variable of type Boolean should start with a verb (e.g. foundCarrier, isPresent).

#### Ordering

* Using directives must be placed within a namespace ([SA1200](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1200)).
* Elements at the file root level or within a namespace must be positioned in the following order ([SA1201](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1201)):

1. extern alias directives
2. using directives
3. namespaces
4. delegates
5. enumerations
6. interfaces
7. structs
8. classes

Within a class, struct or interface, the following positioning must be applied:

1. fields
2. constructors
3. finalizers (destructors)
4. delegates
5. events
6. enumerations
7. interfaces
8. properties
9. indexers
10. methods
11. structs
12. classes

* Elements must be ordered by accessibility ([SA1202](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1202)). Adjacent elements of the same type must adhere to the following positioning:

1. public
2. internal
3. protected internal
4. protected
5. private

* Constants must appear before fields ([SA1203](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1203)).
* Static elements must appear before instance elements ([SA1204](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1204)).
* Partial elements must declare access ([SA1205](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1205)).
* Declaration keywords must adhere to the following ordering scheme ([SA1206](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1206)):

1. access modifiers
2. static
3. all other keywords

* The keyword *protected* must be positioned before the keyword *internal* when declaring a protected internal C# element ([SA1207](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1207)).
* Property accessors must adhere to the following ordering scheme ([SA1212](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1212)):

1. get accessors
2. set accessors

* Event accessors must adhere to the following ordering scheme ([SA1213](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1213)):

1. add accessor
2. remove accessor

* Static read-only elements must appear before static non-read-only elements ([SA1214](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1214)).
* Instance read-only elements must appear before instance non-read-only elements ([SA1215](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1215)).
* System using directives must be placed before other using directives ([SA1208](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1208)).
* Using alias directives must be placed after other using directives ([SA1209](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1209)).
* Using directives must be ordered alphabetically by namespace ([SA1210](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1210)).
* Using alias directives must be ordered alphabetically by alias name ([SA1211](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1211)).

Note: In Visual Studio, you can easily sort the using directives by positioning the cursor on a using directive in the code editor and selecting *Organize Usings* > *Sort Usings* from the context menu. In case you want to also remove unnecessary using directives, select *Organize Usings* > *Remove and Sort Usings* from the context menu.

In order to let Visual Studio put System directives first when sorting namespaces, go to *Tools* > *Options*. In the list on the left, go to *Text Editor* > *C#*> *Advanced* and select the check box *Place 'System' directives first when sorting usings*.

#### Layout

In order to improve readability, consider the following layout guidelines:

* Allman bracing style should be used, e.g. the curly brace associated with a control statement must be placed on the next line and must be indented to the same level as the control statement. Statements within the braces are indented to the next level ([SA1500](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1500), [SA1501](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1501), [SA1502](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1502)).

if (slKeysToDelete.Count > 0)

{

protocol.DeleteRow(16000, slKeysToDelete.ToArray());

}

* In general, curly braces should never be considered optional ([SA1503](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1503)). For statements where curly braces are optional, these may only be omitted if the statement is written on a single line.

Note: This rule is not enabled in the StyleCop configuration file.

* The accessors within a property, indexer or event must either all be written on a single line or all be written on multiple lines ([SA1504](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1504)).
* Opening curly braces must not be preceded or followed by a blank line ([SA1509](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1509), [SA1505](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1505)) and closing curly braces must not be preceded by a blank line ([SA1508](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1508)). Closing curly braces must be followed by a blank line, unless the next line is a closing curly brace ([SA1513](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1513)).
* An XML documentation header above an element must not be followed by a blank line ([SA1506](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1506)) and must be preceded by a blank line ([SA1514](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1514)).
* Code must not contain multiple blank lines in a row ([SA1507](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1507)).
* Chained statement blocks must not be preceded by a blank line ([SA1510](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1510)).
* The while footer of a do-while statement must not be separated from the statement by a blank line ([SA1511](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1511)).
* Adjacent C# elements must be separated by a blank line ([SA1516](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1516)), e.g. a blank line should be provided between property definitions and method definitions.
* Code must not contain blank lines at the start and end of the file ([SA1517](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1517), [SA1518](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1518)).

#### Spacing

* A semicolon should always be followed by a single space, unless it is the last character on the line. A semicolon should never be preceded by any whitespace, unless it is the first character on the line ([SA1002](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1002)).
* Preprocessor keywords must not be preceded by space ([SA1006](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1006)).
* Operator keyword must be followed by a space ([SA1007](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1007)).
* Member access symbols must be spaced correctly ([SA1019](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1019)). A member access symbol should not have whitespace on either side, unless it is the first character on the line.
* Increment/decrement symbols must be spaced correctly ([SA1020](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1020)). There should be no whitespace between the increment or decrement symbol and the item that is being incremented or decremented.
* Positive/negative signs must be spaced correctly ([SA1022](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1022), [SA1021](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1021)). A positive/negative sign should always be preceded by a single space, unless it comes after an opening square bracket or a parenthesis, or it is the first character on the line. A positive/negative sign should never be followed by whitespace and should never be the last character on a line.
* Dereference and access-of symbols must be spaced correctly ([SA1023](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1023)).
* Colons must be spaced correctly ([SA1024](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1024)).
  + A colon appearing within an element declaration must always have a single space on either side, unless it is the first or last character on the line.
  + When the colon comes at the end of a label or case statement, it must always be followed by whitespace or be the last character on the line, but it should never be preceded by whitespace.
  + When a colon is used within a conditional statement, it must always contain a single space on either side, unless the colon is the first or last character on the line.
* Code must not contain multiple whitespace characters in a row ([SA1025](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1025)).
* Within an implicitly typed new array allocation, there should not be any space between the new keyword and the opening array bracket ([SA1026](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1026)).
* Do not split null conditional operators ([SA1029](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1029)). The null conditional symbol (‘?.’ or ‘?[‘) should not be separated by a new row, space or comment.

#### Readability

* Do not prefix calls with *base* unless a local implementation exists ([SA1100](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1100)).
* There should be only one declaration per line.
* Code must not contain empty statements ([SA1106](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1106)).
* There should be only one statement per line ([SA1107](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1107)).
* Method parameter placement/indexer:
  + The opening parenthesis must be on the declaration line ([SA1110](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1110)).
  + The closing parenthesis must be on the line of the last parameter ([SA1111](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1111)).
  + In case a method or indexer does not take any parameters, the closing parenthesis must be on the line of the opening parenthesis ([SA1112](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1112)).
  + A comma must be on the same line as the previous parameter ([SA1113](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1113)).
  + A parameter list must follow the declaration ([SA1114](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1114)).
  + A parameter must follow a comma ([SA1115](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1115)). Ensure that the parameter begins on the same line as the previous comma, or on the next line.
  + Split parameters must start on the line after the declaration ([SA1116](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1116)).
  + Parameters must all be on the same line or each on a separate line ([SA1117](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1117)).
  + A parameter must not span multiple lines ([SA1118](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1118)).

It is preferred to write a method call in a single line. When this line becomes too long, you can either:

* + - Introduce one or more temporary variables that hold the parameter values and pass those as parameters (this is the preferred option), or
    - Let every parameter start on a new line.
* Use String.Empty for empty strings ([SA1122](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1122)).
* The use of C# type aliases should be favored over the use of the predefined types in the System namespace (e.g. use string sUserName instead of String sUserName), except when calling static members (e.g. use String.IsNullOrEmpty(sResponse) instead of string.IsNullOrEmpty(sResponse)) ([SA1121](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1121)).

Note: As StyleCop rule SA1121 also expects to use C# type aliases for static members, this rule has not been enabled in the StyleCop configuration file.

* Use shorthand notation for nullable types ([SA1125](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1125)).
* Query clauses:
  + A query clause must follow the previous clause ([SA1102](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1102)).
  + Query clauses must either each be on a separate line or all be on one line ([SA1103](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1103)).
  + A query clause must begin on a new line when the previous clause spans multiple lines ([SA1104](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1104)).
  + Query clauses spanning multiple lines must begin on a new line ([SA1105](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1105)).
* Comments must contain text ([SA1120](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1120)).

#### Maintainability

* Access modifiers must be explicitly declared ([SA1400](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1400)).
* Fields must be private ([SA1401](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1401)).
* Statements must not use unnecessary parentheses ([SA1119](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1119)).
* Arithmetic and conditional expressions must declare precedence ([SA1407](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1407), [SA1408](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1408)).
* Unnecessary parentheses in attribute constructors must not be used ([SA1411](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1411)).
* Code analysis suppression must have justification ([SA1404](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1404)).
* A debug assert or fail must include message text ([SA1405](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1405), [SA1406](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1406)).
* Remove unnecessary code ([SA1409](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1409)).
* Remove delegate parentheses when possible ([SA1410](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1410)).

#### XML Documentation

|  |  |
| --- | --- |
| **NOTE:** | For more information about XML documentation, refer to:   * [Document your code with XML comments](https://docs.microsoft.com/en-us/dotnet/csharp/codedoc) * [How to: Insert XML comments for documentation generation](https://docs.microsoft.com/en-us/visualstudio/ide/reference/generate-xml-documentation-comments?view=vs-2019) * [XML documentation comments (C# Programming Guide)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/xmldoc/xml-documentation-comments) |

* When useful, (partial) elements should be documented ([SA1600](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1600), [SA1601](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1601)).

Note: These rules are not enabled in the StyleCop configuration file. However, for manager protocols or protocols implementing complex functionality, elements should be documented when useful (e.g. shared QActions).

* Documentation must contain valid XML ([SA1603](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1603)).
* (Partial) element documentation must have a summary ([SA1604](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1604), [SA1605](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1605)).
* (Partial) element documentation must have summary text ([SA1606](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1606), [SA1607](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1607)) that differs from the default summary text ([SA1608](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1608)).
* Documentation lines must begin with a single space ([SA1004](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1004)).
* Element parameters
  + Element parameters must be documented ([SA1611](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1611)).
  + Element parameter documentation must match element parameters ([SA1612](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1612)).
  + Element parameter documentation must declare the parameter name ([SA1613](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1613)).
  + Element parameter documentation must have text ([SA1614](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1614)).
* The element return value must be documented ([SA1615](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1615), [SA1616](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1616)).
* A void return type must not be documented ([SA1617](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1617)).
* Generic type parameters
  + Generic type parameters must be documented ([SA1618](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1618), [SA1619](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1619)).
  + Generic type parameter documentation must match the type parameters ([SA1620](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1620)).
  + Generic type parameter documentation must declare a parameter name ([SA1621](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1621)).
  + Generic type parameter documentation must have text ([SA1622](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1622)).
* Property summary documentation must match accessors ([SA1623](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1623)). The property’s summary text must begin with wording describing the types of accessors exposed within the property. If the property contains only a get accessor, the summary must begin with the word “Gets”. If the property contains only a set accessor, the summary must begin with the word “Sets”. If the property exposes both a get and set accessor, the summary text must begin with “Gets or sets”.

If the property returns a Boolean value, an additional rule is applied. The summary text for Boolean properties must contain the words “Gets a value indicating whether”, “Sets a value indicating whether”, or “Gets or sets a value indicating whether”.

* Property summary documentation must omit set accessors with restricted access ([SA1624](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1624)). When an accessor within the property is given an access level that is more limited than the access level of the property, this accessor should be omitted from the summary documentation.
* Element documentation must not be copied and pasted ([SA1625](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1625)).
* Documentation text:
  + Documentation text must not be empty ([SA1627](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1627)).
  + Documentation text must begin with an uppercase letter and end with a period ([SA1628](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1628), [SA1629](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1629)).
  + Documentation text must contain whitespace ([SA1630](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1630)).
  + Documentation text must meet the character percentage ([SA1631](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1631)). This rule is calculated by counting the number of alphabetic characters and numbers within the documentation text, and comparing it against the number of symbols and other non-alphabetic characters.
  + Documentation text must meet the minimum character length ([SA1632](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1632)).
* Constructor summary text must begin with “Initializes a new instance of the {class name} class.” ([SA1642](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1642)).
* Destructor summary text must begin with “Finalizes an instance of the {class name} class.” ([SA1643](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1643)).
* The documentation header must not contain blank lines ([SA1644](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1644)).
* An included Documentation XPath must contain a valid path ([SA1646](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1646)).
* An include node must contain a valid path ([SA1647](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1647)).
* <inheritdoc> must be used with an inheriting class ([SA1648](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1648)).
* Element documentation must be spelled correctly ([SA1650](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1650)).

#### Commenting

* Unless a comment is very short, comments should always start on a new line.
* A single-line comment must be preceded by a blank line ([SA1515](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1515)) and must not be followed by a blank line ([SA1512](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1512)).
* A space should be inserted between the comment delimiter (i.e. “//”) and the comment text ([SA1005](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1005)).
* For comments spanning multiple lines, the use of single-line syntax “//” is favored over using “/\* \*/”.
* Creation of formatted comment boxes (e.g. through the use of asterisks) should be avoided.
* Comments should be indented at the same level as the code they describe.
* Single-line comments must not use XML documentation style slashes ([SA1626](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1626)). Comments beginning with three slashes are reserved for XML documentation headers.
* Block statements must not contain embedded comments ([SA1108](https://github.com/Visual-Stylecop/Visual-StyleCop/wiki/SA1108)).
* When commenting out lines of code, begin the comment with four slashes to differentiate it from normal comments.
* A comment should begin with an uppercase letter and end with a period.
* Comments should not contain spelling mistakes.
* If meaningful variable and method names are used, the code should already be readable. Therefore, comments should only be added in case these will further clarify the code.

## Administrative Metadata

### Comment Metadata

* A Comment section must be available at the beginning of the protocol file containing the following metadata, which must be compliant with the protocol registration on DCP.

1. Copyright
2. Ownership

### Protocol Metadata

* The following protocol metadata tags must be defined and compliant with the protocol registration on DCP.

1. Protocol Name
2. Protocol Description
3. Protocol Version
4. Protocol Integration ID (DMS-DRV-XXXX)
5. Protocol Provider
6. Protocol Vendor
7. Protocol Vendor OID (1.3.6.1.4.1.8813.2.XX)
8. Protocol Device OID
9. Protocol Element Type
10. Protocol Version History

## User Interface

### Element Wizard

#### Default Settings

* The port settings must contain as many default settings as possible and must be as strict as possible. This also includes disabling options that are not applicable (e.g. disabling irrelevant options: in case of a HTTP driver, we should not be able to select type of port as “Serial”).

The default bus address “ByPassProxy” must be used in drivers of type HTTP.

#### Connection Names

* If there is only one connection for a specific type, the name needs to have the following format, where the second option can (optionally) be used to add more info about the goal of the connection (e.g. XXX = Traps, XXX = Events, XXX = Alarms, etc.):
  + “IP Connection" or "IP Connection - XXX": For drivers that support TCP and/or UDP.
  + "HTTP Connection" or "HTTP Connection - XXX"
  + "SNMP Connection" or "SNMP Connection - XXX"
  + "Serial Connection" or "Serial Connection - XXX": For drivers that only support the physical serial port, i.e. driver connections of type serial that do not support TCP or UDP.
* If there is more than one connection for a specific type, the name needs have the following format, where XXX is used to distinguish between the connections (e.g. XXX = Redundant, XXX = Redundant 2, XXX = Backup, XXX = Traps, XXX = Events, etc.):
  + "IP Connection - XXX"
  + "HTTP Connection - XXX"
  + Etc.

### Protocol Pages

#### Page Layout

* By default, a maximum of two columns is allowed on a page.

#### Look and Feel

* Vendors typically organize their devices into different product lines or series. The same look and feel must be provided for all drivers for devices belonging to the same product line.

#### General Page

* Each protocol must have a page with the name “General”, containing some general parameters (e.g. the device name, the device type, the firmware version, etc.), which must be used as the default page.

#### Core Functionality Pages

* A protocol is typically developed for a specific device (e.g. demodulator, integrated receiver/decoder (IRD), etc.). A protocol must reflect the functionality of the device by including a page for each functional block of the device. For example, a protocol for an integrated receiver/decoder should include a page for the receiver and a page for the decoder.

These pages must contain the core parameters of the device, which should be a combination of status parameters and important configuration parameters. The order of these parameters should reflect the internal workflow of the device (e.g. in case of a decoder, first configure the frequency, then the modulation, etc.).

#### Web Interface Page

* In case the device has a web user interface, this must be added to the protocol as an embedded object on its own page, so that it can be viewed through the element card. The web interface page must be the last page and it must be preceded by a page separator. The name of this page should be “Web Interface”.

### Displayed Text

All displayed textual items, e.g. parameter descriptions, button values, discrete values, tooltips, element wizard text etc. that will be displayed in DataMiner must be user-friendly and adhere to the following rules:

#### Title Case

When title case should be used (e.g. parameter descriptions), every word should be capitalized, except for:

* Articles (a, an, the)
* Coordinating conjunctions (and, but, for, etc.)
* Prepositions of three letters or less (at, by, to, etc.)
* Words representing a unit of measure, brand, etc.: For these the official capitalization must be applied.

#### Brand and Product Names

* Brand or product names must use their official capitalization.

#### Acronyms and Initialisms

* Acronyms and initialisms must always be written with capital letters according to the defining standards or specifications. Acronyms and initialisms are types of abbreviations that consist of the first letters of other words and are used as a word in themselves (The difference between acronyms and initialisms is that acronyms can be pronounced as a word, whereas initialisms are pronounced as the separate letters. For example, BIOS and CORBA are acronyms, IP and DVB-2 are initialisms). In case the web interface of a device displays such abbreviations, verify their meaning and correct capitalization.

E.g. the following table gives an overview of some frequently occurring acronyms and initialisms for transport streams:

|  |  |
| --- | --- |
| Abbreviation | Description |
| CA | Conditional Access |
| CAT | Conditional Access Table |
| CBR | Constant Bitrate |
| CC | Continuity Counter |
| DTS | Decoding Time Stamp |
| FEC | Forward Error Correction |
| LTW | Legal Time Window |
| MPTS | Multi Program Transport Stream |
| NIT | Network Information Table |
| PAT | Program Association Table |
| PCR | Program Clock Reference |
| PES | Packetized Elementary Stream |
| PID | Packet Identifier |
| PMT | Program Map Table |
| PSI | Program Specific Information |
| PTS | Presentation Time Stamp |
| SPTS | Single Program Transport Stream |
| STC | System Time Clock |
| TEI | Transport Error Indicator |
| TS | Transport Stream |

|  |  |
| --- | --- |
| **Note**: | Units of measure are subject to specific casing. |

### Parameters

#### Parameter Description and Values

* Values like ‘True/False’ must be avoided. Rephrase the parameter description if necessary. For example, instead of defining a parameter with description “Manual Mode” and possible values ‘True” and “False”, it is considered better practice to use the description “Switch Mode” and values “Automatic” and “Manual”.

#### Unit of Measure

* If applicable, a unit of measure must be provided, which must correspond to one described in the DataMiner Protocol Development Library.

#### Value Range

* If applicable, a value range must be provided (Display.Range) for displayed parameters (read and write).

#### Date and Time Values

* Parameters used to hold a time, date or datetime value should be implemented using the “date”, “time” or “datetime” option.

#### Exceptional States

* In case a device can return exceptional states (e.g. “N/A”, “Unknown”, etc.), these should be implemented as “exceptions” or “others” and have the disabled state.

#### Tooltips

* All displayed parameters must have a meaningful smart tag (tooltip).

#### Octet Values

* Octet values should be implemented as bit rate.

#### Error Counter Values

* Error Counters should be made available as a rate value.

### Tables

#### Column Names

* The names of column parameters of a table must all start with the name of the table. This makes it possible to quickly see which parameters belong to a table in the protocol tree.

#### Column Descriptions

* In case a table has a column that has the same description as a column defined in another table, add the table description (or an abbreviation of the table description) at the end of the column description, in parentheses. For example, for tables with the names “Services” and “Streams”, the following column descriptions can be used: “State (Services)” and “State (Streams)”.

#### Column Header Options

* Since DataMiner version 8.5, columns of type “number” that support alarming will by default show the sum of all values in the column. In case this default behavior is irrelevant, it must be either disabled (e.g. disableHeaderSum) or replaced by a more suitable header (e.g. enableHeaderMax). Also, note that histograms and heat maps should be disabled in case these are not relevant.

### Buttons

* The width of a button must be explicitly defined in the protocol. A minimum width of 110 is advised. Only use a larger value when necessary. However, keep the width of buttons as uniform as possible (e.g. in case there are two buttons and for one button a width of 120 is required, it is better to also set the width of the other button to 120).

### Toggle Buttons

* A writeable parameter with only two possible values (e.g. “On/Off” or “Enabled/Disabled’) must be implemented as a toggle button if the second value is obvious when you read the first value. In case the second value is not obvious when you read the first, a drop-down box must be used.

### Page Buttons

#### Page Button Dimensions

* The width of a page button must be explicitly defined in the protocol. A minimum width of 110 is advised. Only use a larger value when necessary. However, keep the width of page buttons as uniform as possible (e.g. in case there are two page buttons and for one page button a width of 120 is required, it is better to also set the width of the other page button to 120).

#### Page Button Labels

* The label of a page button must always end with an ellipsis (‘…’), so a distinction can be made between a page button and a regular button. The ellipsis must immediately follow the label, so no space is allowed before the ellipsis (e.g. “Configure…” instead of “Configure …”).

#### Page Button Content

* Page buttons are used to display a limited amount of data. A pop-up window opened by means of a page button should contain either data of less importance or data that should remain visible while users are working on other pages.
* A pop-up page opened by means of a page button should not contain page buttons.

### Progress Bars

* In case an operation can take a long time to complete, the use of a progress bar should be considered.

## Monitoring

### Trending

* By default, all parameters must allow trending, except for parameters used for internal logic, table definition parameters, index parameters, foreign key parameters, etc.
* Make sure no redundant trending can be set on parameters.
* For parameters of type string, when trending is supported, the trend type should be set to “last”. (cf. PROTOCOL.PARAMS.PARAM.DISPLAY.TRENDING.TYPE section in the DataMiner Protocol Development Guide.)

### Alarming

* By default, all parameters must support alarming, except for parameters used for internal logic, table definition parameters, index parameters, foreign key parameters, etc.
* Make sure no redundant alarms can be set on parameters.
* If applicable, default values should be provided for the different severity levels.
* Do not implement parameters used for the configuration of alarm thresholds just because the user interface of the device provides these. DataMiner can create those values by itself by means of an alarm template.

## Protocol

### API

#### Device API

* The implemented communication protocol in a protocol must correspond to the documented device API as defined by the vendor. An SNMP protocol is defined by a MIB file and the format must match (e.g. SNMP types). A vendor-specific protocol needs to be implemented as described in the API (e.g. a specific CRC).

#### DataMiner API

* The DataMiner API must be used as defined in the DataMiner Protocol Development Guide and the most efficient documented methods must be used.

### Communication

* Communication with the device must not wait until protocol timeout time. E.g. invalid structure of serial responses.

### Logic

#### Operation Duration

* No action must take longer than 15 minutes, because after 15 minutes a Run-Time Error (RTE) will be generated. Keep this in mind when creating e.g. QActions and groups.

|  |  |
| --- | --- |
| **Note**: | After 7.5 min, a half-open RTE message is logged in the SLWatchdog2.txt log file. |

#### Operation Execution Delay

* Make sure that there are no large delays between operations. E.g. a button set should be executed right away. Because of this, the content of a group must be limited to a maximum of 10 pairs or SNMP parameters. In case SNMP tables are used in groups, each table should be included in a different group containing only this table.

#### Value Update Verification

* In case a protocol enables performing sets on a device (e.g. changing a configuration parameter), the protocol must verify that the set succeeded by retrieving the value again (e.g. when using SNMP, a get should always be performed after the set to ensure the set succeeded).

#### Protocol Threads

* Consider using separate protocol threads in cases where this could be beneficial, e.g. when multiple different types of functionality are implemented and can run independently of each other. However, make sure to verify that the implementation using multiple protocol threads actually performs better.

### Timers

#### Number of Timers

* The number of timers in a protocol must be kept as low as possible, as every timer in a protocol creates an additional thread.

#### Default Timer Speeds

* Unless otherwise specified by the TAM, the following default timer speeds must be used:

1. Alarm and status parameters: 10 seconds.
2. Configuration parameters: 1 minute. (Set the execution frequency of the included groups when Data Display is opened to 30 seconds by setting the dataDisplay attribute to a value of 30000.)
3. Static data: 1 hour.
4. Each timer must have a default interval of 75 (ms).

#### Timer Speed Restrictions

* The speed of timers must be carefully chosen. For example, suppose a timer triggers every 10 seconds and initiates an operation that requires 20 seconds to complete. This can have a negative impact on performance (SLScripting, SLProtocol). The timer interval must be set to a value allowing every group included in the timer to complete before the timer triggers again. This can be verified with SLNet Client Pendingcalls and the element logging. Note also that a timer interval cannot exceed 24 hours, since a thread in SLWatchdog cannot be registered for longer than that.
* The set timer time must not exceed 24 days.

#### Starting/Stopping Timers

* The use of conditions must always be favored over starting, stopping and restarting timers, but a condition on a group is favored over a condition on a timer, since the timer performs a regular check on that condition to see if it becomes true. In case an action is used to start or stop a timer, a timer that is already started must not be started again, as this could lead to issues.

#### Timer Content

* The last group in a timer should be a group of type “poll”, “poll trigger” or “poll action”, to guarantee that the timer does not start again while not all groups of the timer have been executed yet.

### DVE

#### DVE Names

* The name of a DVE protocol should be constructed as follows: Mother Protocol Name – Product Name.

#### DVE Export Rules

* Verify that an export rule removes the table name suffixes when parameters are exported as standalone parameters in a DVE element. For example, when the main DVE element has a column named “Admin Status (IF Table)”, the description in the DVE child element should be “Admin Status”.

#### DVE Child Element Deletion

* DVE protocols must not delete DVE child elements automatically by default. [(Ref. 3.5.6.4 Data Handling)](#_Data_Handling)

### Tables

#### Primary Keys

* A table always contains a column where each row has a unique value identifying the row, i.e. the column containing the primary keys. This must be the first column in the table.
* In the DataMiner MySQL database, the first 100 bytes of a row key must be unique. Otherwise, unexpected behavior can occur, e.g. some rows will be visible in a table but not stored in the database. UTF-8-character encoding is used by default, which encodes a character in as many bytes as required, with a maximum of 4. Characters from the ASCII character set are encoded using a single byte, so the first 100 characters must be unique. In a worst-case scenario, all characters take up 4 bytes, meaning that the first 25 characters must be unique.
* Numeric keys are more efficient than long non-numeric keys. In order to improve performance, the use of an auto increment key should be considered instead of long non-numeric keys, if applicable.
* Backslashes are not allowed in the primary keys because MySQL does not allow these.

#### Display Keys

* In case the values in the primary key column are not user-friendly, another column or a combination of other columns must be used to construct a more user-friendly label that is also unique for every row. In case a combination of columns is used, an additional column must be added to the table that contains the result of this combination (see Naming, NamingFormat, DisplayColumn, DisplayKey in the DataMiner Protocol Development Guide). A column of type “displaykey” must be the last column of the table.
* Naming should be favored over using the displayColumn attribute. The implementation of displayColumn has an impact on trend data in the database, as the data is stored with a relation between the key and the value of displayColumn. Every time the displayColumn value changes, the database will be updated.
* Existing drivers that use displayColumn should not be changed to use naming (as this will result in loss of data) unless this change is required for a specific reason.

#### Key Column Description

* The column containing the unique user-friendly label that is used to allow users to identify a row (i.e. the display key in case the primary key is not user-friendly) must have “[IDX]” included in the description. In case the table name is added between parentheses for additional context, “[IDX]” should be positioned before the table name, e.g. “Name [IDX] (Devices)”.

#### Data Handling

* Depending on the type of data a table will hold, different scenarios are possible as to what should be done with the table data when the table is updated.

In many cases, a table should only hold the entries that are currently available. If so, old rows that are no longer present will be removed. For example, SNMP and WMI tables will by default only need to display the entries that currently exist, and old rows will be removed.

|  |  |
| --- | --- |
| **Note**: | For tables filled in by a QAction, code is necessary to remove the old data, e.g. via FillArray, RemoveRow. |

Previous data comparison

Sometimes users need to know about items that are no longer present compared to a previous polling of the table data (e.g. missing transport streams, services, processes, DVEs, etc.). In this case, the items that are no longer present must not be automatically removed. The table should indicate which entries are no longer available.

This should be implemented as follows:

* + Add a PageButton “[Table Name] Config…” above the table that contains:
    - A toggle button “[Table Name] Auto Clear” (On/Off, default value Off). When the table is updated, check this status and remove the row(s) if needed.
    - A button “Remove All [Content Type]” to remove all missing or old entries.
    - Table:
      * A column containing buttons to remove a missing entry.

Infinitely growing amount of data

Another possible scenario is that entries keep being added to a table (e.g. data pushed from an external third-party component to DataMiner, such as SNMP traps). In this case, the protocol must provide a means to automatically limit the size of the table. However, care must be taken to not remove entries that are still valid and of interest to the user (e.g. only remove cleared traps, old information events, etc.).

Implementation based on maximum number of rows and/or maximum time to keep:

* + Add a PageButton “[Table Name] Config…” above the table that contains:
    - A toggle button “Max Table Size” (On/Off, default: On)
    - A parameter to define the maximum allowed number of rows. (Provide a well-chosen default value).
    - A parameter to define the maximum time to keep. (Provide a well-chosen default value.)

|  |  |
| --- | --- |
| **Note**: | In this case, the table must contain a column that holds the time (in descending sort order). |

Performance of inter-process calls

FillArray or SetColumn calls should try to stay below 25 000 cells in 1 set.

As a rule of thumb, for tables with 20 to 25 columns, avoid setting more than 1000 rows at a time. If your table is larger, split this up into multiple sets per 1000 rows to improve performance.

Note that this is a general guideline that has exceptions depending on your situation. For example, if the content of a cell is very large (such as 20 mb strings), your code will perform better if you add more separation in the calls than stated above.

#### Integrity Constraints

* Integrity constraints defined on tables (e.g. use of foreign keys) should be respected. For example, suppose table B contains a foreign key to table A. When removing a row from table A, make sure that rows in table B that link to this row are removed first.

#### Preserve State Option

* The use of the “preserve state” option on table columns should be avoided, as this option requires significantly more processing.

### Parameters

#### Saving Parameters

* Parameters must only be saved when necessary. When a parameter is saved, it is saved in the database, leading to additional processing and memory usage.

#### Loading Parameters in SLElement

* RTDisplay must only be set to “true” when necessary. Every parameter of type “read” that has RTDisplay set to “true” will be loaded into the SLElement process. In case a parameter is not positioned on any page and RTDisplay is set to “true” in order to make it externally accessible, provide a comment explaining why this is necessary. In addition, set the onAppLevel attribute to “true”.

### QActions

#### Functionality

* QActions should only be used when no other protocol constructs are available to implement the desired functionality.

#### Maintainability

* In order to keep code maintainable, code that is reused in different QActions must be moved to a general QAction. The ID of this general QAction should be 1.

|  |  |
| --- | --- |
| **Note**: | * Use the option “precompile”, so the general QAction is compiled immediately. When using the code from QAction 1 in another QAction, the value of the dllImport attribute should have the following format: [ProtocolName].[ProtocolVersion].QAction.1.dll; * When a general QAction is used, consider using different namespaces to create logical groups of related functionality. |

#### Robustness

* Whenever a specific format is expected in a QAction variable, verify the format before using it. If the variable does not match the format, add a log entry.

#### Conditional Execution

* Conditions should be used instead of a conditional statement at the beginning of a QAction entry point that checks if the QAction should be executed.

#### Localization

* As DataMiner is used across the world, regional settings of DataMiner servers can differ, i.e. different regional cultures or locales can be in use. This means that, among others, specific formatting of dates and numbers is applied. When data is requested or sent (in particular strings that contain culture-dependent information such as numbers and dates), this must be taken into account to ensure the correct results.

For example, suppose a DataMiner Agent requests a value from a device and the device sends back the following string: “10.05MHz”. In this case, the device uses a dot as the decimal symbol. However, it is possible that the DataMiner Agent has regional settings applied where a comma is used as the decimal symbol. In this case, converting the numeric part of this string using Convert.ToDouble("10.05") would not result in the correct conversion, as the current local culture is used, and the current local culture assumes a comma is used as the decimal symbol. Instead, e.g. Double.TryParse("10.05", NumberStyles.Float, CultureInfo.InvariantCulture, out signalStrength) should be used.

For the same reason, when a command containing a numeric value as string is created from a QAction, conversion using culture info is necessary as otherwise the command string could be formatted incorrectly. E.g. Convert.ToString(signalStrength) could result in “10,05” where the device would expect “10.05”. Therefore, additional culture info is necessary when the conversion is done: Convert.ToString(signalStrength, CultureInfo.InvariantCulture);.

|  |  |
| --- | --- |
| **Note**: | When the value of a protocol parameter containing a value of type double is retrieved, culture info is not required. You can use the ToDouble method, i.e. the overload that requires only the value object as parameter (defined in the Convert class (namespace System)), e.g. double signalStrength = Convert.ToDouble(protocol.SingalStrength);. |

#### Inter-Process Communication

The number of interactions between SLScripting, the process executing the QAction, and other processes should be limited to an absolute minimum, as these consume a considerable amount of resources. Also limit the number of interactions per second.

It can occur that a QAction needs to perform an operation that requires some time to complete (e.g. a DMS call). In case another call is needed to determine if the operation has finished, you must avoid continuously repeating the call that checks the operation status without leaving time between the calls (for example, by including a Thread.Sleep call in the loop).

Typically, QActions mainly interact with the SLProtocol process using an instance of the SLProtocol (or SLProtocolExt) class. As each protocol object call results in inter-process communication between the SLScripting and SLProtocol processes, consider the following:

* Use the GetParameters and SetParameters methods instead of multiple GetParameter and SetParameter method calls to get and set multiple parameters, respectively.
* The implementation with the best performance should be used when retrieving or updating tables. A QAction implementation that retrieves or sets column(s) is generally preferred over a QAction that triggers on every row of a table (row=true).
* Protocol calls inside a loop should be avoided as much as possible.

In order to improve readability (and type safety) of code in QActions, the following guidelines should be considered:

* In case a wrapper method is available for a NotifyProtocol call, the use of the wrapper method is favored.
* Usage of the SLProtocolExt class is favored over the SLProtocol class, as in many situations this allows you to write more readable code.
* With DataMiner versions prior to DataMiner 9.0, the use of the GetKeys method to retrieve the primary keys (NotifyProtocol.KeyType.Index) should be avoided, as up to DataMiner 9.0 the implementation to retrieve the primary keys is based on the SLElement process (an NT\_GET\_INDEXES call is executed, which retrieves both the primary keys and the display keys.). As it is possible that there is a delay in SLElement (e.g. due to a large number of calls that it is processing), it is possible that the returned data is not up to date.

Since DataMiner 9.0, the implementation of the GetKeys method has been updated. Retrieving the primary keys no longer involves the SLElement process. However, note that obtaining the display keys (NotifyProtocol.KeyType.DisplayKey) is still based on SLElement.

|  |  |
| --- | --- |
| **Note**: | * The implementation of the Keys property of the QActionTable class also makes use of the GetKeys method. Therefore, the use of the Keys property should also be avoided prior to DataMiner 9.0. * The ClearAllKeys() method uses the NT\_GET\_INDEXES call in all DataMiner versions, so for this reason the use of this method should always be avoided. |

#### Exception Handling

* In case a call is performed that could throw an exception, exception handling must be provided.
* In case a *try* block allocates resources, resource cleanup must be provided in a *finally* block

|  |  |
| --- | --- |
| **Note**: | The usage of a using clause is preferred |

* When an IDisposable object is used, the object should be declared and instantiated in a using statement. The using statement ensures that the Dispose method is called (whether an exception occurred or not).

|  |  |
| --- | --- |
| **Note**: | The object is read-only and cannot be modified or reassigned. |

* Every method serving as an entry point in a QAction should contain exception handling, so that it cannot throw exceptions. This is because otherwise a memory leak can occur in the SLScripting process.
* The ToString method of the exception object should be used when exception-related information is logged (instead of the Message property).

protocol.Log("QA" + protocol.QActionID + "|Method Name|" + e.ToString(), LogType.Error, LogLevel.NoLogging);

#### Logging

* When logging messages are generated, the QAction ID and method name should be included:

protocol.Log("QA" + protocol.QActionID + "|Method Name|Message", LogType.Error, LogLevel.NoLogging);

* Logging messages that should only be generated when debugging a protocol can be surrounded by preprocessor directives as follows:

#if debug

protocol.Log("QA" + protocol.QActionID + "|Method Name|Message", LogType.Error, LogLevel.NoLogging);

#endif

* Only implement logging when required, e.g. logging exceptions.

#### Multi-threading

* In some situations, parallel programming can be considered to optimize performance and decrease execution time. The .NET Framework 4 (and above) provides extensive support for parallel programming. Consider the following when implementing multi-threading in a protocol:
  + Verify that the parallel implementation is in fact running faster through performance measurements, especially when using locks.
  + Exploiting parallel programming makes available CPU resources for other processes decrease. Therefore, we must avoid that the SLScripting process consumes all CPU resources, as otherwise all other DataMiner processes will slow down or freeze, which can cause a DataMiner crash.
  + Threads initiated in a QAction must be finished before the QAction finishes. Having a thread still running while the QAction that initiated it is already finished must be avoided for the following reasons:
    - Many SLProtocol methods cannot be used anymore.
    - The SLWatchDog process will not notice if items are stuck (no RTE notice).
    - Investigating issues related to threads outlasting QActions is very difficult.
    - An unhandled exception in a thread that outlasts the QAction that initiated it causes the SLScripting process to crash.

|  |  |
| --- | --- |
| **See also**: | <https://msdn.microsoft.com/en-us/library/dd460693(v=vs.110).aspx> |

* DataMiner takes care of locking when performing calls to update local or remote parameters. However, a QAction may contain critical sections of code. In that case, locks must be provided to ensure that a thread does not enter the critical section while another thread is in it. For example, suppose a row is retrieved from a table and the QAction implements logic to update some cells of the row. At a later point, a call is made to update the table with the new row values. In this case, it is important to provide a lock around the code to prevent the row being removed in the meantime. This data retrieval, manipulation and update sequence is a common scenario where locking is required.

#### Thread.Sleep

* When Thread.Sleep is used, the time to sleep should be at least 15 ms and it should be a multiple of 15 ms. This is considered best practice because the default System Timer Resolution (15 ms) will force values below 15 ms to use 15 ms instead. Actual values will always be a multiple of 15 ms. E.g. Sleep 16 will actually be 30 ms; Sleep 5 ms will be 15 ms. The default timer resolution can be changed by multimedia applications (e.g. Cube) up to 1 ms, which then influences any defined Thread.Sleep. A value of 3 ms can then actually be 3 ms, which means that the actual measured impact on a platform may change in comparison with the default timer resolution of 15 ms.

Using Thread.Sleep in a protocol should be exceptional. A good reason must be present to introduce a Sleep. Verify therefore the usefulness and that it does not e.g. introduce long execution times.

#### StyleCop

* StyleCop does not indicate any warnings. For an overview of the StyleCop rules that need to be adhered to, refer to Section “[C# Code Conventions](#_C#_Code_Conventions)”.

|  |  |
| --- | --- |
| **See also**: | <https://github.com/Visual-Stylecop/Visual-StyleCop/wiki> |

### .NET Recommendations

This section contains a number of guidelines to keep in mind when developing code. For additional guidelines related to design decisions, refer to <https://msdn.microsoft.com/en-us/library/ms229042(v=vs.110).aspx>.

#### Microbenchmarking

When code optimization is performed, measurements should be taken (microbenchmarking) in order to be able to confirm that the optimization is indeed improving performance and to get an estimation of the obtained performance gain. To perform measurements, the Stopwatch class (System.Diagnostics) can be used.

Some things to consider when microbenchmarking:

* During microbenchmarking, no other processes should be running (to avoid other processes impacting the microbenchmark).
* When measuring code that takes little time to execute, multiple iterations should be executed.
* The test could be optimized away by the language or JIT compiler (e.g. in Release mode).
* Often, the first measurement result is discarded, as this measurement can be influenced by startup costs.
* The measurement time overhead should be subtracted from the total measured time.

#### String vs. StringBuilder

String objects are immutable (i.e. they cannot be modified after they have been created). This means that all String methods and C# operators, while appearing to modify a string, actually return the result in a new string object.

Therefore, when a String object will be modified frequently, the use of a StringBuilder instance should be considered. The MSDN documentation for the StringBuilder class states the following:   
Although the StringBuilder class generally offers better performance than the String class, String should not automatically be replaced with StringBuilder whenever a string manipulation is needed. Performance depends on the size of the string, the amount of memory to be allocated for the new string, the system on which the application is executing, and the type of operation. You should be prepared to test your app to determine whether StringBuilder actually offers a significant performance improvement.

|  |  |
| --- | --- |
| **See also**: | <https://msdn.microsoft.com/En-Us/library/system.text.stringbuilder(v=vs.110).aspx> |

Consider using the String class under the following conditions:

* When the number of changes that the application will make to a string is small. In these cases, StringBuilder might offer negligible or no performance improvement over String.
* When a fixed number of concatenation operations are performed, particularly with string literals. In this case, the compiler might combine the concatenation operations into a single operation.
* When extensive search operations have to be performed while building the string. The StringBuilder class lacks search methods such as IndexOf or StartsWith. For these operations, the StringBuilder object has to be converted to a String, and this can negate the performance benefit from using StringBuilder. For more information, see the section “[Searching the text in a StringBuilder object](https://msdn.microsoft.com/En-Us/library/system.text.stringbuilder(v=vs.110).aspx#Searching)” in the above-mentioned MSDN documentation.

Consider using the StringBuilder class under the following conditions:

* When the application is expected to make an unknown number of changes to a string at design time (e.g. when a loop is used to concatenate a random number of strings that contain user input).
* When the application is expected to make a significant number of changes to a string.

Also note that string literals can be split in order to improve readability. The compiler will concatenate the parts into a single string, so there is no run-time performance cost.

|  |  |
| --- | --- |
| **Note:** | Often, code is needed in a protocol to generate a string that represents a list of items that are separated by a character or string (e.g. when using a parameter that uses the dependencyId attribute). An alternative implementation is to collect all items in a list (List<T>) and then use the String.Join method (e.g. string sDevices = string.Join(";", lsDevices);). |

#### Conditional Statements

* To avoid exceptions and increase performance by skipping unnecessary comparisons, use the conditional-AND operator (“&&”) instead of the &-operator, and the conditional-OR operator (“||”) instead of the |-operator. This will avoid unnecessary evaluations (also known as short-circuit evaluation).
* In a conditional-AND statement, the condition that is most likely to fail should be put first. In a conditional-OR statement, the condition that is most likely to succeed should be put first.
* In selection statements (i.e. if-else and in some cases switch statements) the data distribution should be kept in mind. For example, when implementing a switch statement with three possible cases, e.g. A, B and C, while it is known that C will occur most of the time, performance can be optimized by first checking for C.

#### Loops

* The amount of code inside a loop should be kept as small as possible. Code that is not needed in a loop should be moved out of the loop.
* Avoid calling properties inside a loop.
* Break out of the loop as soon as possible (using the break keyword).
* Go to the next iteration as soon as possible (using the continue keyword).

#### Collections

* The .NET Framework defines many different types of collections, each designed for a specific purpose. Choose your collection class carefully by considering the characteristics of each collection type. The following table gives an overview of the characteristics of some commonly used collection types.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Collection | Description | Insertion | | Removal | | Retrieval/Lookup | | Immediate access |
| List<T> | Represents a strongly typed list of objects that can be accessed by index. | Add(T) | O(1)-O(n)\* | Remove(T) | O(n) | Item[Int32]  IndexOf(T) | O(1)  O(n) | Index |
| LinkedList<T> | Represents a doubly linked list. | AddFirst(T)  AddLast(T)  AddAfter | O(1)  O(1)  O(1) | RemoveFirst  RemoveLast  Remove(T) | O(1)  O(1)  O(n) | Find(T) | O(n) | No |
| HashSet<T> | Represents a set of values. | Add(T) | O(1)-O(n)\* | Remove(T) | O(1) | Contains(T) | O(1) | Key |
| Dictionary<TKey,TValue> | Represents a collection of keys and values. | Add | O(1)-O(n)\* | Remove(TKey) | O(1)\*\* | Item[TKey] | O(1)\*\* | Key |
| Stack | Represents a last-in, first-out (LIFO) collection. | Push(T) | O(1)-O(n)\* | Pop() | O(1) | - | - | No |
| Queue | Represents a first-in, first-out (FIFO) collection. | Enqueue(Obj) | O(1)-O(n)\* | Dequeue() | O(1) | - | - | No |
| SortedList | Represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index. | Add | O(log n)\*\*\* | Remove(Obj)  RemoveAt(Int32) | O(n)  O(n) | Item[Object] | O(log n) | Key |
| SortedDictionary<TKey,TValue> | Represents a collection of key/value pairs that are sorted on the key. | Add | O(log n) | Remove(TKey) | O(log n) | Item[Key] | O(log n) | Key |
| SortedSet<T> | Represents a collection of objects that is maintained in sorted order. | Add(T) | O(log n) | Remove(T) | O(log n) | Contains(T) | O(ln n) | Key |

\* If [Count](https://msdn.microsoft.com/en-us/library/27b47ht3(v=vs.110).aspx) is less than [Capacity](https://msdn.microsoft.com/en-us/library/y52x03h2(v=vs.110).aspx), this method is an O(1) operation. If the capacity needs to be increased to accommodate the new element, this method becomes an O(n) operation, where n is [Count](https://msdn.microsoft.com/en-us/library/27b47ht3(v=vs.110).aspx).

\*\* Approaches O(1) operation, includes key lookup overhead.

\*\*\* This method is an O(n) operation for unsorted data, where n is [Count](https://msdn.microsoft.com/en-us/library/system.collections.sortedlist.count(v=vs.110).aspx). It is an O(log n) operation if the new element is added at the end of the list. If insertion causes a resize, the operation is O(n).

* In some cases, a combination of collections can be used to obtain optimal performance (at the cost of increased memory usage). For example, suppose there are two files: a file containing 1,000,000 keys and a large CSV file (1.5 GB) where only the lines that match one of the keys in the other file should be processed. First, the key file is read and stored in a collection. Then the CSV file is read. For each line in the CSV file, the Contains() method on the collection instance is used to check if it needs to be processed.

When the keys are stored using an instance of List<string>, reading the keys file takes 6 s. The lookup, however, takes considerably more time (> 20 min).

When an instance of Stack<string> is used, it takes 200 ms to read out the keys file. However, the lookup still takes a long time (> 20 min).

When an instance of HashSet<string> is used, it takes 26 s to read out the keys file. The lookup takes 26 s.

Performance can be optimized by using a combination of collections: The key file is first loaded in an instance of Stack<string>. This object is then used to construct an instance of a HashSet<string>. This takes 400 ms and the lookup takes 26 s. This results in a total processing time of 26.4s. However, note that this performance increase comes at a cost of increased memory usage.

* If Count exceeds Capacity while elements are added, the capacity is increased by automatically reallocating the internal array before copying the old elements and adding the new elements. Collections can be instantiated with an initial capacity to reduce the number of reallocations. Also, when multiple items need to be added to a collection, the AddRange method is favored over multiple Add method calls.
* Although the ArrayList class and List<T> class have similar functionality, the List<T> class performs better in most cases. Moreover, List<T> is type safe. In case a heterogeneous collection needs to stored, an instance of List<object> should be used instead of an instance of the ArrayList class. In case a homogeneous collection needs to be stored, a strongly typed list (List<T>) should be used.

MSDN states the following regarding performance considerations:

*“In deciding whether to use the List<T> or [ArrayList](https://msdn.microsoft.com/en-us/library/system.collections.arraylist(v=vs.110).aspx) class, both of which have similar functionality, remember that the List<T> class performs better in most cases and is type safe. If a reference type is used for type T of the List<T> class, the behavior of the two classes is identical. However, if a value type is used for type T, you need to consider implementation and boxing issues.*

*If a value type is used for type***T***, the compiler generates an implementation of the****List<T>****class specifically for that value type. That means a list element of a****List<T>****object does not have to be boxed before the element can be used, and after about 500 list elements are created, the memory saved not boxing list elements is greater than the memory used to generate the class implementation.*

*Make certain the value type used for type*T*implements the IEquatable<T> generic interface. If not, methods such as Contains must call the Object.Equals(Object) method, which boxes the affected list element. If the value type implements the IComparable interface and you own the source code, also implement the IComparable<T> generic interface to prevent the BinarySearch and Sort methods from boxing list elements. If you do not own the source code, pass an IComparer<T> object to the BinarySearch and Sort methods.*

*It is to your advantage to use the type-specific implementation of the****List<T>****class instead of using the ArrayList class or writing a strongly typed wrapper collection yourself. The reason is your implementation must do what the .NET Framework does for you already, and the common language runtime can share Microsoft intermediate language code and metadata, which your implementation cannot.”*

|  |  |
| --- | --- |
| **See also**: | * <https://msdn.microsoft.com/en-us/library/system.collections.arraylist(v=vs.110).aspx> * <https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx> |

#### LINQ

* LINQ allows you to write compact and readable code. Though in some cases an alternative implementation can perform even better, LINQ is generally preferred as it will typically result in more readable and maintainable code than the alternative implementation. Only use other specific implementations when these outperform LIQ and the performance of the implementation has a major impact on the overall performance.
* A query is a set of instructions on how to retrieve and organize data. To execute the query, a call to its GetEnumerator method is required (deferred execution). This call is made when a foreach loop is used to iterate over the elements. To evaluate a query and store its results without executing a foreach loop, a ToList, ToArray, ToDictionary, or ToLookup should be called on the query variable.

When looping over an IENumerable, values are received from the source data one at a time. This can be very positive for the total memory usage in case the data source is large and the data first needs to be filtered before it is used.

XDocument xDocument = XDocument.Load(@"C:\...\Response.xml");

var loopData = xDocument.Descendants().Where(p => (p.Attribute("id")).Contains("i8"));

foreach (var v in loopData)

{

// Do something.

string x = v.TryGetAttribute("id");

int leng = x.Length;

}

This is very efficient as the variable loopData will not contain a separate list with all filtered values. This means that the data that is already loaded in the xDoc will not be saved somewhere else. This also means that the line below should in theory have no noticeable effect on memory or CPU when loopData is never called:

var loopData = xDocument.Descendants().Where(p => (p.Attribute("id")).Contains("i8"));

However, this can have a negative impact on execution time in case the IEnumerable is used multiple times. The following example illustrates this:

IEnumerable<string> allKeysInXML = xDocument.Descendants("MonitorGroup").Select(p => p.Attribute("DISPLAYNAME")).Distinct();

var keysToRemove = keysInOriginal.Except(allKeysInXML);

var keysToAdd = allKeysInXML.Except(keysInOriginal);

The excerpt above retrieves all the keys in the original that are not in the XML document and all the keys in the XML document that are not present in the original. The keys in the XML document are retrieved using LINQ. However, the IEnumerable is used twice in the statements following the LINQ query. Therefore, the following filter gets executed twice:

Descendants("MonitorGroup").Select(p => p.Attribute("DISPLAYNAME")).Distinct();

When a ToList() call is added, the filter will only be executed once increasing performance (and, as already mentioned, increased memory usage).

## Validation

### General

* Verify that parameter get and set operations succeed. Verify that each configurable parameter of the device is set correctly.
* Try to verify the parameter value using the web interface of the device or another tool. If this is not possible, try to verify the format of these settings using Stream Viewer for a locally created element.
* In order to verify the communication with the device, you must verify displayed parameters that are not initialized. Parameters should not have exceptions by default to avoid being not initialized.
* No errors or issues must occur when the device is not reachable and communication should still be initiated.

### Spell Checker

* The Visual Studio plugin “Spell Checker” must be used to verify that the protocol is using US spelling and all valid errors should be fixed.

### DIS Validator

* The DIS validator (insider release build) needs to be executed and all notifications must be handled.

### Stream Viewer

* No problems may be indicated in Stream Viewer.

### Log Files

* The element log file or other log files do not contain problems related to an element running the implemented protocol.

### Alarm Console

* The Alarm Console does not indicate any problems related to an element running the implemented protocol.

## Points of Attention

### Skyline Driver Passport Platform

* A DataMiner Test package has been created and scheduled on the Driver Passport Platform in order to validate performance metrics as part of the Code Review process or when deemed useful.

### Dojo Use Cases

* For more information related to the procedure, refer to <https://skylinebe.sharepoint.com/sites/DeployandAccelerate/SitePages/Dojo-connector-use-cases.aspx> (only accessible for Skyline employees).

# Abbreviations

|  |  |
| --- | --- |
| ASCII | American Standard Code for Information Interchange |
| API | Application Programming Interface |
| CPU | Central Processing Unit |
| CSV | Comma-Separated Values |
| DLL | Dynamic-link Library |
| DMA | DataMiner Agent |
| DMS | DataMiner System |
| DVE | Dynamic Virtual Element |
| HTTP | Hypertext Transfer Protocol |
| IRD | Integrated Receiver/Decoder |
| LINQ | Language Integrated Query |
| RN | Release Note |
| RTE | Run-time Exception |
| SNMP | Simple Network Management Protocol |
| UTF-8 | Unicode Transformation Formats-8 |
| WMI | Windows Management Instrumentation |
| XML | eXtensible Markup Language |