SIEMENS

Introduction 1

References 2

Code examples 3

Startdrive

SINAMICS
Functions for Startdrive

Programming and Operating Manual

2018

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
GERMANY

℗ 01/2019 Subject to change

Copyright © Siemens AG 2017 - 2018.
All rights reserved

Table of contents

1 Introduction 5
2 References 7
2.1 AddressComposition 7
2.2 AddressContext 8
2.3 AddressIoType 9
2.4 ConfigurationEntry 10
2.5 DriveDomainFunctions 11
2.6 DriveObject 12
2.7 DriveObjectActivation 13
2.8 DriveObjectContainer 14
2.9 DriveObjectTypeHandler 15
2.10 DriveParameter 16
2.11 DriveParameterComposition 18
2.12 EncoderConfiguration 19
2.13 HardwareProjection 20
2.14 MotorConfiguration 21
2.15 OnlineDriveObject 23
2.16 OnlineDriveObjectContainer 24
2.17 StartDriveDownloadCheckConfiguration 25
2.18 SafetyTelegram 26
2.19 Telegram 27
2.20 TelegramComposition 28
2.21 TelegramType 30
2.22 TorqueTelegram 31
3 Code examples 33
3.1 Determining the activation status 33
3.2 Executing drive functions 34
3.3 Creating a drive unit 35
3.4 Creating a drive component 36
3.5 Determining a drive object 37
3.6 Determining the drive object type 38

3.7 Reading and writing BICO parameters 39
3.8 Download 40
3.9 Carrying out the first steps in Startdrive 42
3.10 Defining the encoder type 43
3.11 Configuring devices 50
3.12 Creating a component for a drive component (S120 only) 51
3.13 Defining the motor type and motor configuration 53
3.14 Reading and writing parameters 56
3.15 Reading and writing parameters online 59
3.16 Saving the parameterization 60
3.17 Using Safety Integrated telegrams 61
3.18 Inserting and extending telegrams 62
3.19 Using torque telegrams 63
3.20 Restoring factory settings 64

Introduction

1

Using TIA Portal Openness, you automate the engineering workflow and control the TIA Portal from the program that you generated yourself.

In this help document, you can find a lot of information and code examples for this program that you generate yourself. You can also configure your own programs and apply them for the TIA Portal "Startdrive" application.

Before you configure your own program for Startdrive from the subsequently listed code example, please carefully observe the general information relating to Openness, which you can find under the following keywords in this help document:

- Requirements for TIA Portal Openness
- Install TIA Portal Openness
- Access the TIA Portal
- TIA Portal Openness object model
- Programming steps

Using TypeIdentifiers

Each version of any Startdrive component comprises a unique number, which is called the TypeIdentifier . In the Openness program code, you can use these TypeIdentifiers in order to uniquely identify and name a component.

In Startdrive, the display of the TypeIdentifier is optional, and deactivated as default.

Activating the display of the TypeIdentifier in Startdrive

1. In the Startdrive project view, select menu "Options > Settings".
   The "Settings" configuration area opens.
2. Select "Hardware configuration" in the secondary navigation.
3. Activate the option "Activate display of the Type Identifier for devices and modules".
   The display of the TypeIdentifier is now active.

Reading out the TypeIdentifier in Startdrive

In a Startdrive project, the TypeIdentifier can be read out at the following locations when the display is active:

- For all components (in the inspector window):
- For Control Units (when creating a drive device)

To read out the TypeIdentifier for a component in the inspector window, proceed as follows:

1. In the device view of the Startdrive project, double-click on the required component.
   The inspector window opens. The active component version is shown in the list.
2. The associated TypeIdentifiers are listed in the outer right-hand column.
   Example: `OrderNumber:6SL3131-7TE23-6Axx`
   Copy this TypeIdentifier to your Openness application.

Proceed as follows to read out the TypeIdentifier for a Control Unit:

1. In the Startdrive project navigation, double click on "Add new device".
   The dialog with the same name opens.
2. Select the required control module from the selection.
   The TypeIdentifier (under the article number and firmware number) for the corresponding Control Unit is displayed to the right in the detailed display.
   Example: `OrderNumber:6SL3040-1MA01-0Axx/V5.2/S120`
3. Copy this TypeIdentifier to your Openness application.

References

2

2.1 AddressComposition

AddressComposition

The `AddressComposition` class represents the address of a telegram.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public sealed class AddressComposition
```

The following table describes the **properties** of the class:

Name	Data type	Description
`IoType`	AddressIoType (Page 9)	Returns information on the type of the address.
`Context`	AddressContext (Page 8)	Returns information on the context of the address.
`StartAddress`	`Int32`	Returns the start address of the telegram or specifies it.
`Length`	`Int32`	Returns the length of the telegram.

See also

TelegramType (Page 30)

2.2 AddressContext

AddressContext

The Enum AddressContext contains information on the context of the address.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public enum AddressContext
```

The following table describes the **enum entries**:

Name	Description
`AddressContext.None`	No context has been found for the address
`AddressContext.Device`	Context is a device address
`AddressContext.Head`	Context is a head address

2.3 AddressIoType

AddressIoType

The Enum AddressIoType contains information on the type of the address.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public enum AddressIoType
```

The following table describes the **enum entries**:

Name	Description
`AddressIoType.None`	The IO type cannot be used
`AddressIoType.Input`	Type is an input address
`AddressIoType.Output`	Type is an output address
`AddressIoType.Diagnosis`	Type is a diagnosis address
`AddressIoType.Substitute`	Type is a substitute address

2.4 ConfigurationEntry

ConfigurationEntry

Class `ConfigurationEntry` is used to save parameter data, which can be determined from the `ConfigurationEntryCompositions` of a motor or encoder configuration.

The following table describes the **properties** of the class:

Name	Data type	Description
`EnumValueList`	`IDictionary<it, string>`	Returns a list with possible values of the enum parameter.
`MaxValue`	`object`	Maximum value of the `ConfigurationEntry`.
`MinValue`	`object`	Minimum value of the `ConfigurationEntry`.
`Name`	`string`	Name of the `ConfigurationEntry`.
`Number`	`int`	Numerical representation of the name for `ConfigurationEntry`.
`Description`	`string`	Description of the `ConfigurationEntry`.
`Parent`	`IEngineeringObject`	Engineering Object model-parent element of this object.
`Unit`	`string`	Unit of the `ConfigurationEntry`.
`Value`	`object`	Value of the `ConfigurationEntry`.

2.5 DriveDomainFunctions

DriveDomainFunctions

Class `DriveDomainFunctions` is used to restore the factory settings or the backup of the RAM content to the ROM.

It can only be applied to an `OnlineDriveFunctionInterface` object.

For G120 drives, the `DriveDomainFunctions` object can only be accessed if the Power Module is connected to the device. Otherwise, null or an exception is returned.

The following table describes the **methods** of the class:

Name	Description
`PerformFactoryReset`	This method is responsible for restoring the factory settings.
`PerformRAMtoROMCopyAllDriveObject`	The date of all drive objects is written from the RAM to the memory card/ hard disk.

The following table describes the **properties** of the class:

Name	Data type	Description
`Parent`	`IEngineeringObject`	Engineering Object model-parent element of this object.

2.6 DriveObject

DriveObject

The `DriveObject` class allows access to the drive object. Access to the drive parameters or the telegram is possible via the drive object, for example.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public sealed class DriveObject
```

The following table describes the **properties** of the class:

Name	Data type	Description
`Parameters`	DriveParameter-Composition (Page 18)	Returns a list with the available parameters of the drive object.
`Telegrams`	TelegramComposition (Page 28)	Returns a list with the available telegrams of the drive object. The list can be changed with class `TelegramComposition`.

See also

Determining a drive object (Page 37)

2.7 DriveObjectActivation

DriveObjectActivation

Class `DriveObjectActivation` is used to activate the modules or determine the module status. It can be applied to `DriveObjectFunctions` from `DriveFunctionInterface` or `OnlineDriveFunctionInterface`.

The following table describes the **methods** of the class:

Name	Description
`ChangeActivationState(DriveObjectActivationState)`	Changes the activation status of the drive object. Returns false if the operation cannot be completed. Possible status values: • Deactivate • Activate • DeactivateAndNotPresent

The following table describes the **properties** of the class:

Name	Data type	Description
`ActivationState`	`DriveObjectActivationState`	Returns the actual status value.
`IsActive`	`Boolean`	For an active drive object, returns true.

2.8 DriveObjectContainer

DriveObjectContainer

The `DriveObjectContainer` is a service of the drive object (`DeviceItem`) for the current device (`Device`).

The following table describes the **navigators** of the `DriveObjectContainer`:

Name	Data type	Description
`DriveObjects`	DriveObjectComposition (Page 18)	Returns a list with the available drive objects. The drive objects allow access to the drive parameters and telegrams.

2.9 DriveObjectTypeHandler

DriveObjectTypeHandler

Class `DriveObjectTypeHandler` is used to switch over the drive object type for every drive object and to determine the drive object type - as well as all possible drive object types of the actual drive object. It can only be applied to `DriveFunctionInterface`.

The following table describes the **methods** of the class:

Name	Description
`ChangeDriveObjectType((target)DriveObjectType)`	Changes the actual type of the drive object to a new type that can be selected. Returns false if the operation cannot be completed.

The following table describes the **properties** of the class:

Name	Data type	Description
`PossibleDriveObjectTypes`	`DriveObjectTypeComposition`	The use of `targetDriveObjectType` results in an exception on the actual drive object.
`CurrentDriveObjectType`	`DriveObjectType`	Indicates the currently assigned drive object type.

2.10 DriveParameter

DriveParameter

The `DriveParameter` class allows access to a drive parameter. Not all drive parameters are available for access via Openness.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public sealed class DriveParameter
```

The following table describes the **properties** of the class:

Name	Data type	Description
`ArrayIndex`	`Int32`	Returns the index of an array parameter. Value range: 0-7FFF The index is -1 for parameters without array **Example** `p108[4].15` `par.ArrayIndex` produces 4
`ArrayLength`	`Int32`	Returns the number of array elements. The index is 0 for parameters without an array

`Bits`	DriveParameter-Composition (Page 18)	Returns an `DriveParameter` object for one bit of the parameter.
		In this way, for example, the value or the name of the bit parameter can be read from a bit parameter.
		Example
		`DriveParameter param133 = cu.Parameters.Find(133, 0);`
		`DriveParameter param133Bit1 = param133.Bits[1];`
		`String paramName = param133Bit1.Name;`

`EnumValueList`	`IDictionary<int, string>`	Returns a list with possible values of the enum parameter. For example `<1, [1] Quick commissioning>` `null`, if the parameter is not a parameter of the `Enum` type.
`MaxValue`	`Object`	Returns the maximum value for the currently selected unit.
`MinValue`	`Object`	Returns the minimum value for the currently selected unit.
`Name`	`string`	Returns the name of the parameter. E.g. "`p108[0].2`"
`ParameterText`	`string`	Returns the text of the short description for the parameter.
`Number`	`Int32`	Returns the number of the parameter. **Example** `p108[0].2` The return value is 108

Name	Data type	Description
`Unit`	`string`	Returns the unit of the parameter as text.
`Value`	`Object`	Returns the offline/online value of the parameter or writes a value onto the parameter. If the event of write errors, an `EngineeringTargetInvocationException` is triggered. **Examples** • `P2080Bit6.Value = 0;` • `P2080Bit6.Value = cu.Parameters.Find("r19");` **BICO source** The parameters of a BICO source can only be read **BICO signal sinks** Possible values are 0, 1 or a `DriveParameter` object. A `DriveParameter` object is returned when the BICO signal sink is connected to a different parameter. See also example Reading and writing BICO parameters (Page 39).

See also

Reading and writing parameters (Page 56)

2.11 DriveParameterComposition

DriveParameterComposition

The `DriveParameterComposition` class allows access to parameters of the drive. Not all drive parameters are approved for access via Openness.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public sealed class DriveParameterComposition
```

The following table describes the **methods** of the class:

Name	Description
`Find(string)`	Returns the DriveParameter (Page 16) object for which a search is being made via the name. `null` if the parameter is not found **Example** `Find("P108[1]");`
`Find(UInt16, Int32)`	Returns the DriveParameter (Page 16) object for which a search is being made via the parameter index and array index. `null` if the parameter is not found **Examples** • `cu.Find(108, 1);` • `cu.Find(51, -1);`

`WriteParameters(IEnumerable<string>, IEnumerable<string>, bool)`	Writes values in parameters.
	With the `ignoreErrors = true` setting, an attempt is made to write all values in the event of an error and an `EngineeringTargetInvocationException` is triggered at the end.
	For SINAMICS G drives, parameter values can only be written with a configured Power Module (PM).
	Example
	`List<string> names = new List<string>();`
	`List<string> values = new List<string>();`
	`names.add("p300[0])");`
	`values.add("17");`
	`names.add("p5391[0])");`
	`values.add("20");`
	
	`cu.WriteParameters(names, values, true);`

2.12 EncoderConfiguration

EncoderConfiguration

Class `EncoderConfiguration` saves data of non-Siemens encoders.

- The user must populate the `ConfigurationEntryComposition` object.
- Object `RequiredConfigurationEntries` must also be populated.

Namespace: `Siemens.Engineering.MC.Drives`
`Siemens.Engineering.MC.Drives.DFI`
`Siemens.Engineering.MC.Drives.Enum`

Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **properties** of the class:

Name	Data type	Description
`RequiredConfigurationEntries`	`ConfigurationEntryComposition`	Accessible configuration entries of the motor configuration.
`Parent`	`IEngineeringObject`	Engineering Object model-parent element of this object.

2.13 HardwareProjection

Class `HardwareProjection` is responsible for commissioning the motor and the encoder. The object can be found for `DriveFunctionInterface` and `OnlineDriveFunctionInterface`.

For G120 drives, motors and encoders can be configured both online and offline. On the other hand, for S120 drives, configuration is only possible offline.

For G120 drive devices, the `HardwareProjection` object can only be accessed if the Power Module is inserted in the drive device. Otherwise, when calling functions `HardwareProjection`, a null or an exception is returned.

For an offline configuration, use the hardware configuration of the `DriveFunctionInterface`. For an online configuration, use the hardware configuration of the `OnlineDriveFunctionInterface`.

Namespace: `Siemens.Engineering.MC.Drives`
`Siemens.Engineering.MC.Drives.DFI`
`Siemens.Engineering.MC.Drives.Enums`

Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **methods** of the class:

Name	Description
`SetMotorType(MotorType type, ushort driveDataSet)`	Sets the motor type at the Control Unit (only for G120).
`GetCurrentMotorConfiguration( ushort driveDataSet)`	Depending on the data set number of the drive, determines the currently existing configuration area.
`ProjectMotorConfiguration(MotorConfiguration motConfig, ushort driveDataSet)`	Configures the motor configuration of a drive device depending on the data set number of the drive.
`SetEncoder(EncoderType type, EncoderInterface interfaceType, AbsoluteIncrementalFlag absIncFlag, RotaryLinearFlag rotLinFlag, ushort encDataSet)`	Sets the encoder at the Control Unit (only for G120).
`GetCurrentEncoderConfiguration(ushort encDataSet)`	Depending on the data set number of the encoder, determines the currently existing configuration area.
`ProjectEncoderConfiguration(EncoderConfiguration encConfig, ushort encDataSet)`	Configures the motor configuration of a drive device depending on the data set number of the encoder.

The following table describes the **properties** of the class:

Name	Data type	Description
`Parent`	`IEngineeringObject`	Engineering Object model-parent element of this object.

2.14 MotorConfiguration

MotorConfiguration

Class `MotorConfiguration` is responsible for commissioning motors and encoders.

Namespace: `Siemens.Engineering.MC.Drives`
`Siemens.Engineering.MC.Drives.DFI`
`Siemens.Engineering.MC.Drives.Enums`

Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

Response for Siemens motors:

Action `SetMotortype` must be called to set the motor type. This action comprises the Enum `MotorType` and a number, which represents the `DriveDatasetNumber`.

The following table describes the enums:

Enum name	Description
`NoMotor`	0: No motor
`InductionMotor`	1: Induction motor
`SynchronousMotor`	2: Synchronous motor
`NoCodeNumber1LE1InductionMotor`	10: 1LE1 induction motor (not a code number)
`NoCodeNumber1LG6InductionMotor`	13: 1LG6 induction motor (not a code number)
`NoCodeNumber1xx1SIMOTICSFDInductionMotor`	14: 1xx1 SIMOTICS FD induction motor (not a code number)
`NoCodeNumber1LA7InductionMotorNoCodeNumber`	17: 1LA7 induction motor (not a code number)
`MotorSeriesNumber1LA81PQ8StandardInduction`	18: 1LA8 / 1PQ8 standard induction motor series
`NoCodeNumber1LA9InductionMotor`	19: 1LA9 induction motor (not a code number)

Response for non-Siemens motors:

Class `MotorConfiguration` is used to save data of non-Siemens motors. It contains 2 objects `ConfigurationEntryComposition`, which, when commissioning, must be populated with the corresponding motor data. Object `RequiredConfigurationEntries` must also be populated. Object `OptionalConfigurationEntries` does not have to be populated.

The following table describes the **properties** of the class:

Name	Data type	Description
`RequiredConfigurationEntries`	`ConfigurationEntryComposition`	Accessible configuration entries of this motor configuration.
`OptionalConfigurationEntries`	`ConfigurationEntryComposition`	Accessible configuration entries of this motor configuration.
`Parent`	`IEngineeringObject`	Engineering Object model-parent element of this object.

2.15 OnlineDriveObject

OnlineDriveObject

The `OnlineDriveObject` class allows online access to the drive object. Drive parameters can be accessed via the drive object.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public sealed class OnlineDriveObject
```

The following table describes the **properties** of the class:

Name	Data type	Description
`Parameters`	DriveParameter-Composition (Page 18)	Returns a list with the available parameters of the online drive object. `null`, if the mode is "Offline". In offline mode, an exception is triggered when a method is called or for a write access to a parameter.

See also

Determining a drive object (Page 37)

Reading and writing parameters (Page 56)

Reading and writing parameters online (Page 59)

2.16 OnlineDriveObjectContainer

OnlineDriveObjectContainer

The `OnlineDriveObjectContainer` is a service of the drive object (`DeviceItem`) for the current device (`Device`).

The following table describes the **navigators** of the `OnlineDriveObjectContainer`:

Name	Data type	Description
`OnlineDriveObjects`	`OnlineDriveObjectComposition`	Returns a list with the available online drive objects (OnlineDriveObject (Page 23)). The drive objects allow access to the drive parameters.

2.17 StartDriveDownloadCheckConfiguration

StartDriveDownloadCheckConfiguration

Class `StartDriveDownloadCheckConfiguration` is derived from the class `DownloadCheckConfiguration` and has the same properties.
Class `DownloadCheckConfiguration` is described in the standard Openness help.

The class provides the configuration settings via checkboxes from the user.

The following table describes the **properties** of the class:

Name	Data type	Description
`Checked`	`bool`	Returns the current setting of the configuration or activates/ deactivates the configuration.

See also

Download (Page 40)

2.18 SafetyTelegram

SafetyTelegram

Class `SafetyTelegram` stands for the telegram of the drive object.
Exception `EngineeringTargetInvocationException` is displayed for errors in the write attributes.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **properties** of the class:

Name	Data type	Description
`TelegramNumber`	`Int32`	Number of the main telegram. Free telegrams are designated with a value of 999.
`Type`	TelegramType (Page 30)	Telegram type that is returned as enum "TelegramType".
`Addresses`	AddressComposition (Page 7)	Composition of all addresses of a telegram.
`PKW`	Telegram (Page 27)	Composition of all PKW channels of the telegram. null - if the telegram does not have a PKW part.

2.19 Telegram

Telegram

Class `Telegram` allows access to the structure of a telegram from a drive object.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public sealed class Telegram
```

The following table describes the **properties** of the class:

Name	Data type	Description
`TelegramNumber`	`Int32`	Returns the number of the main telegram or specifies the number. A freely configurable telegram has the number 999.
`Type`	TelegramType (Page 30)	Returns the type of telegram as `Enum TelegramType`.
`Addresses`	AddressComposition (Page 7)	Returns an `AdressComposition` with information on the address.
`PKW`	`Telegram`	Returns channel `PKW` as `Telegram`. `null` if the property is not available A telegram with `PKW` is telegram 353, for example.

The following table describes the **methods** of the class:

Name	Description
`CanChangeTelegram(Int32)`	Returns `true` if the telegram can be changed to the parameterized standard type.
`GetSize(`AddressIoType (Page 9)`)`	Returns the size of the inputs or outputs of the telegram.
`CanChangeSize(`AddressIoType (Page 9)`, Int32, bool)`	Returns `true` if the size of the telegram can be changed as parameterized. Standard telegrams can only be enlarged. The retention of the previous telegram address is taken into account when the option is parameterized with `true`.
`ChangeSize(`AddressIoType (Page 9)`, Int32, bool)`	Returns `true` if the size of the telegram could be changed as parameterized. The retention of the previous telegram address is taken into account when the option is parameterized with `true`.

2.20 TelegramComposition

TelegramComposition

The `TelegramComposition` class allows access to the telegrams of a drive object. The structure of a telegram can be read out via class Telegram (Page 27).

PROFIsafe telegrams are not supported.

Note that referenced objects may become invalid through class `TelegramComposition`. For example, object Telegram (Page 27) becomes invalid after the telegram size is changed.

Namespace: `Siemens.Engineering.MC.Drives`

Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public sealed class TelegramComposition
```

The following table describes the **methods** of the class:

Name	Description
`CanInsertAdditionalTelegram(Int32, Int32)`	Returns `true` if an extension can be created in accordance with the parameterized sizes (input and output sizes).
`CanInsertTorqueTelegram (Int32, telegramNumber)`	When adding a new torque telegram with an already existing telegram number, checks whether this is possible.
`CanInsertSupplementaryTelegram(Int32)`	Returns `true` if a supplementary telegram can be created in accordance with the parameterized telegram number.
`EraseTelegram(TelegramType)`	Deletes a telegram with a known telegram type from the drive object. Returns `true` if the parameterized telegram could be deleted. If a torque telegram is used as type, then object "torqueTelegram" is deleted. If a Safety Integrated telegram is used as type, then object "safetyTelegram" is deleted. Standard telegrams cannot be deleted. In the event of an error, an `EngineeringTargetInvocationException` is initiated.
`Find(TelegramType)`	Returns the Telegram (Page 27) object if it could be found via the parameterized telegram type or Safety Integrated telegram type. `null` if the telegram is not found. If a torque telegram is used as type, then object `torqueTelegram` is returned, assuming that it exists. If a Safety Integrated telegram is used as type, then object `safetyTelegram` is returned, assuming that it exists. **Example** `Telegram telegram = telegrams.Find(TelegramType.MainTelegram);`

Name	Description
`InsertAdditionalTelegram(Int32, Int32)`	Creates an extension for the drive object in accordance with the parameterized sizes and returns `true` if the extension could be inserted. In the event of an error, an `EngineeringTargetInvocationException` is initiated.
`InsertTorqueTelegram (Int32, telegramNumber)`	Adds a new torque telegram with an already existing telegram number to a drive object.
`InsertSafetyTelegram (Int32, telegramNumber)`	Adds a Safety Integrated telegram with its specified telegram number to a drive object.
`InsertSupplementaryTelegram(Int32)`	Creates the supplementary telegram with the parameterized telegram number and returns `true` if the telegram could be inserted. In the event of an error, an `EngineeringTargetInvocationException` is initiated.

2.21 TelegramType

TelegramType

The Enum TelegramType contains predefined telegram types.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **syntax** of the class:

```
public enum TelegramType
```

The following table describes the **enum entries**:

Name	Description
`MainTelegram`	ID of the main telegram
`SupplementaryTelegram`	ID of the supplementary telegram
`AdditionalTelegram`	ID of an extension

2.22 TorqueTelegram

TorqueTelegram

Class `TorqueTelegram` stands for the telegram of the drive object.
Exception `EngineeringTargetInvocationException` is displayed for errors in the write attributes.

Namespace: `Siemens.Engineering.MC.Drives`
Assembly: `Siemens.Engineering.MC.Drives` in `Siemens.Engineering.dll`

The following table describes the **properties** of the class:

Name	Data type	Description
`TelegramNumber`	`Int32`	Telegram number.
`Type`	TelegramType (Page 30)	Telegram type that is returned as enum "TelegramType".
`Addresses`	AddressComposition (Page 7)	Composition of all addresses of a telegram.
`PKW`	Telegram (Page 27)	Composition of all PKW channels of the telegram. null - if the telegram does not have a PKW part.

Code examples 3

3.1 Determining the activation status

The following examples show how you can determine the activation status for S120 drives, either offline or online:

Determining the activation status of an S120 drive offline

```
using Siemens.Engineering.MC.Drives;
DriveFunctionInterface dfi = ...
DriveObjectActivation driveObjectActivation =
dfi.DriveObjectFunctions.DriveObjectActivation;
//driveObjectActivation can be null in case of the actual driveobject does not support
activation.

//change activation state
driveObjectActivation.ChangeActivationState(DriveObjectActivationState.Deactivate);

//get the activation state
DriveObjectActivationState activationState = driveObjectActivation.ActivationState;

//get the Is Active property
bool isActive = driveObjectActivation.IsActive;
```

Determining the activation status of an S120 drive online

```
using Siemens.Engineering.MC.Drives;
OnlineDriveFunctionInterface onlinedfi = ...
DriveObjectActivation driveObjectActivation = onlinedfi.DriveObjectActivation;
//driveObjectActivation can be null in case of the actual driveobject does not support
activation in online.

//change activation state
driveObjectActivation.ChangeActivationState(DriveObjectActivationState.Deactivate);

//get the activation state
DriveObjectActivationState activationState = driveObjectActivation.ActivationState;

//get the IsActive property
bool isActive = driveObjectActivation.IsActive;
```

The following code examples describe the basic procedure for various applications. The code is not necessarily complete or compilable.

3.2 Executing drive functions

The following examples show how you can simply access drive functions, either offline or online:

Executing drive functions offline

```
using Siemens.Engineering.MC.Drives;
DriveObject driveObject = ...
DriveFunctionInterface dfi = driveObject.GetService<DriveFunctionInterface>();

// dfi can be null in case of the actual driveobject does not support it.
```

Executing drive functions online

```
using Siemens.Engineering.MC.Drives;
OnlineDriveObject onlineDriveObject = ...
OnlineDriveFunctionInterface onlineDfi =
onlineDriveObject.GetService<OnlineDriveFunctionInterface>();

// onlineDfi can be null in case of the actual onlineDriveObject
// does not support it or if the device is offline.
```

3.3 Creating a drive unit

You can create a drive unit with method `CreateWithItem()` of collection `Devices`.

The drive units are specified via parameters of the method. The format of the parameters is described in the following.

Creating a G120 drive unit

Format of the parameters for a G120:

```
CreateWithItem(@"OrderNumber: mlfb / FirmwareVersion /",
"NameOfTheDevice", positionNumber)
```

The following example shows how to create a G120 drive unit.

Creating a G120 drive unit

```
TiaPortal portal = new TiaPortal(TiaPortalMode.WithUserInterface);
Project tiaproject= portal.Projects.Open("..."); //The path of the project

Device s120Device =
tiaproject.Devices.CreateWithItem(@"OrderNumber:6SL3246-0BA22-1FA0/4.7.6/"
, "Device_0", null);
```

Creating a S120, S150, MV, G130, G150 drive unit

Format of the parameters for a S120, S150, MV, G130, G150:

```
CreateWithItem(@"OrderNumber: mlfb / FirmwareVersion /
AdditionalTypeIdentifier", "NameOfTheDevice", positionNumber)
```

Possible values for `AdditionalTypeIdentifier`:

- Empty string (e.g. for G120)
- S120
- S150
- MV
- G130
- G150

The following example shows how to create a S120 drive unit.

Creating an S120 drive unit

```
TiaPortal portal = new TiaPortal(TiaPortalMode.WithUserInterface);
Project tiaproject= portal.Projects.Open("..."); //The path of the project

Device s120Device =
tiaproject.Devices.CreateWithItem(@"OrderNumber:6SL3040-1MA01-0Axx/V4.8/
S120", "Device_0", null);
```

3.4 Creating a drive component

You can create a drive component for a drive unit with the `PlugNew()` method of the `Device` object.

The following example shows how to create a drive component.

Creating a Motor Module

```
DeviceItem subModul = sdrDevice.PlugNew(@"OrderNumber:6SL3xxx-xxxxx-xxxx",
"MotorModul", 65535);
```

3.5 Determining a drive object

The following examples show how to determine drive objects offline and online.

Determining an offline drive object

```
using Siemens.Engineering.MC.Drives;
//G device
Project project =  portal.Projects.Open("..."); //Destination folder to
open the project
DeviceItem item = project.Devices[0].Items[0].Items[0];
DriveObject driveObject =
item.GetService<DriveObjectContainer>().DriveObjects[0];

//S device
Project project =  portal.Projects.Open("..."); //Destination folder to
open the project
DeviceItem item = project.Devices[0].Items[0];
DriveObject driveObject =
item.GetService<DriveObjectContainer>().DriveObjects[0];
```

Determining an online drive object

```
using Siemens.Engineering.MC.Drives;
//G device
Project project =  portal.Projects.Open("..."); //Destination folder to
open the project
DeviceItem item = project.Devices[0].Items[0].Items[0];
OnlineDriveObject onlineDriveObject =
item.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];

//S device
Project project =  portal.Projects.Open("..."); //Destination folder to
open the project
DeviceItem item = project.Devices[0].Items[0];
OnlineDriveObject onlineDriveObject =
item.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];
```

3.6 Determining the drive object type

The following examples shows how you can determine the actual drive object type – and the types that can be alternatively set.

Determining drive object types

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;

DriveObject driveObject = ...
DriveFunctionInterface dfi = driveObject.GetService<DriveFunctionInterface>();
DriveObjectTypeHandler driveObjectTypeHandler =
dfi.DriveObjectFunctions.DriveObjectTypeHandler;

// dfi can be null in case of the actual driveobject does not support it.

// driveObjectTypeHandler can be null, if the actual driveObject does not support it.

// Get the possible drive object types on the drive object.
DriveObjectTypeComposition possibleDriveObjectTypes =
driveObjectTypeHandler.PossibleDriveObjectTypes;

// Get the current drive object type on the drive object.
DriveObjectType currentDriveObjectType = driveObjectTypeHandler.CurrentDriveObjectType;

//Call the ChangeDriveObjectType method with the current drive object type.
//The method parameter should be the target drive object type.
driveObjectTypeHandler.ChangeDriveObjectType(possibleDriveObjectTypes[0]);
```

3.7 Reading and writing BICO parameters

The following example shows how to read and write values of BICO parameters. You require a drive object for access.

Reading BICO parameters

```
using Siemens.Engineering.MC.Drives;
DriveParameter bicoSink= driveObject.Parameters.Find("p681");
if(bicoSink!=null)
{
  if(bicoSink.Value is DriveParameter)
  {
    DriveParamter bicoSourceValue = bicoSink.Value as DriveParameter;
    Console.WriteLine("The value of parameter " + bicoSink.Name + ": " +
bicoSource.Name + " " + bicoSource.ParameterText);
  }
  else if (bicoSink.Value == null)
  {
    Console.WriteLine("Value contains an invalid connection or the source
parameter is not accessible via Openness");
  }
  else
  {
    Console.WriteLine("The value of parameter " + bicoSink.Name + ": " +
bicoSink.Value.ToString());
  }
}
```

Writing BICO parameters

```
using Siemens.Engineering.MC.Drives;
DriveParameter bicoSource= driveObject.Parameters.Find("r19");
DriveParameter bicoSink = driveObject.Parameters.Find("p738");
if(bicoSource != null)
{
  try
  {
    bicoSink.Value = bicoSource;
  }
  catch(UserException ex)
  {
    Console.WriteLine("Write failure :" + ex.Message);
  }
}
```

3.8 Download

After starting the download, you must adapt and confirm the configuration settings. The configuration settings are provided as child objects of object `DownloadConfiguration` – and there are three different types:

- `StartDriveDownloadCheckConfiguration`
- `DownloadSelectionConfiguration`
- `DownloadPasswordConfiguration`

The following examples show the evaluation of the different types of configuration settings in the PreDownload Delegate.

Evaluation of the configuration settings after starting the download

```
using Siemens.Engineering.Download;
using Siemens.Engineering.Online;
static void PreDownload(DownloadConfiguration configuration)
{
  Console.WriteLine(configuration.Message);
  StartDriveDownloadCheckConfiguration sdcc = configuration as
StartDriveDownloadCheckConfiguration;
  if (sdcc != null)
  {
    sdcc.Checked = true;
    return;
  }

  DownloadPasswordConfiguration downloadPasswordConfiguration =
configuration as DownloadPasswordConfiguration;

  if (downloadPasswordConfiguration != null)
  {
    SecureString s = new SecureString();
    string passwordText = "password";
    foreach (var str in passwordText)
    {
      s.AppendChar(str);
    }

    downloadPasswordConfiguration.SetPassword(s);
    return;
  }

  DownloadSelectionConfiguration downloadSelectionConfiguration =
configuration as DownloadSelectionConfiguration;

  if (downloadSelectionConfiguration != null)
  {
    downloadSelectionConfiguration.SelectedIndex = 0;
    return;
  }
}
```

The following examples show how to download a project to the device.

Download to the S120 device

```
using Siemens.Engineering.Download;
using Siemens.Engineering.Online;
try
{
  DeviceItem item = ... //device item of the CU (e.g. :
project.Devices[0].Items[0].Items[0])
  DownloadProvider downloadProvider = item.GetService<DownloadProvider>();

  DownloadConfigurationDelegate pre = PreDownload;
  DownloadConfigurationDelegate post = PostDownload;

  ConnectionConfiguration connConfiguration =
downloadProvider.Configuration;
  ConfigurationMode configurationMode = connConfiguration.Modes.Find("PN/
IE");
  ConfigurationPcInterface pcInterface = configurationMode.PcInterfaces[0];
  ConfigurationSubnet subnet = pcInterface.Subnets.Find(/*subnet name*/);
  IConfiguration configuration = subnet.Addresses.Find(/*IP address of the
device*/);
  downloadProvider.Download(configuration, pre, post,
DownloadOptions.Software);
}
catch (Exception e)
{
  Console.WriteLine(e.Message);
}

//configuration handling before download
static void PreDownload(DownloadConfiguration configuration)
{
  Console.WriteLine(configuration.Message);
  StartDriveDownloadCheckConfiguration dcc = configuration as
StartDriveDownloadCheckConfiguration ;
  if (dcc != null)
  {
    dcc.Checked = true;
  }
}

//configuration handling after download
static void PostDownload(DownloadConfiguration configuration)
{
  Console.WriteLine(configuration.Message);
}
```

3.9 Carrying out the first steps in Startdrive

The following example shows how you can localize or generate an active TIA Portal process.

Finding or generating a TIA Portal process

```
using Siemens.Engineering;
// Get the list of the running TIA Portal processes.
IList<TiaPortalProcess> procs = TiaPortal.GetProcesses();
TiaPortal portal;
// When there is at least one running TIA Portal, we will attach to the first from the list.
if (procs.Count != 0)
{
  portal = procs[0].Attach();
}
// When there is no running TIA Portal, we create one.
else
{
  portal = new TiaPortal(TiaPortalMode.WithUserInterface);
}
```

The following example shows how you can localize or generate a Startdrive project.

Finding or generating a Startdrive project

```
using Siemens.Engineering;
Project project;
// When the portal has one project, we save it in a variable.
if (portal.Projects.Count == 1)
{
  project = portal.Projects[0];
}
// When there is no existing project, we create one with a specific path, and the actual time
else
{
  project = portal.Projects.Create(
  new DirectoryInfo(@"C:\Projects\Project_" + DateTime.Now.Ticks),
  DateTime.Now.Ticks.ToString());
}
```

The following example shows how you can determine as to whether a specific Startdrive variant (package and version) is installed.

Determining whether the required Startdrive version is installed

```
using Siemens.Engineering;
if (tiaProcess.InstalledSoftware.Any(sw => sw.Name.Equals("SINAMICS Startdrive Advanced")
&& sw.Version.Equals("V15"))) { Console.WriteLine("Startdrive is available");}
// "V15" is the current startdrive version started at December 2017.
// "V15.1" will be the current startdrive version beginning with December 2018.
// "SINAMICS Startdrive Basic" and "SINAMICS Startdrive Advanced" are the 2 possible
startdrive function packages.
```

3.10 Defining the encoder type

The following example shows how you can set the encoder type or the encoder data set number offline.

Setting the encoder type and/or the encoder data set number offline via the hardware configuration

```
using Siemens.Engineering.MC.Drives
using Siemens.Engineering.MC.Drives.DFI
using Siemens.Engineering.MC.Drives.Enums;

DeviceItem cuDeviceItem = m_Device.DeviceItems[1];
DriveObject cuDriveObject =
cuDeviceItem.GetService<DriveObjectContainer>().DriveObjects[0];

DriveFunctionInterface cuDriveFunctionInterface =
cuDriveObject.GetService<DriveFunctionInterface>();
HardwareProjection hardwareProjection = cuDriveFunctionInterface.HardwareProjection;

// To enable the setting of an EncoderType, the value of p96 should
// be set to 0 (Application class == [0] Expert) if it is present on
// the current G drive.
DriveParameter p96 = cuDriveObject.Parameters.Find("p96");
if (p96 != null)
{
  p96.Value = 0;
}

// Setting Encoder 1 on the drive.
// In case of encoders, we have to set several enums to define an
// encoder type. These enums are: EncoderInterface, EncoderType,
// AbsoluteIncrementalFlag, and RotaryLinearFlag.

// There can be a problem, if the given enum combination is not valid.
// In that case, it has to give back a feedback.

hardwareProjection.SetEncoder(
  EncoderInterface.Terminal,
  EncoderType.HTLTTL,
  AbsoluteIncrementalFlag.Incremental,
  RotaryLinearFlag.Rotary,
  1);

// It is possible to set 2 encoders to a motor, one with encoderNumber == 1
// and the other with encoderNumber == 2
```

The following example shows how you can set the encoder type or the encoder data set number using an online connection.

Setting the encoder type and/or the encoder data set number online via the hardware configuration

```
using Siemens.Engineering.MC.Drives
using Siemens.Engineering.MC.Drives.DFI
using Siemens.Engineering.MC.Drives.Enums;
```

Setting the encoder type and/or the encoder data set number online via the hardware configuration

```
DeviceItem cuDeviceItem = m_Device.DeviceItems[1];
OnlineDriveObject cuOnlineDriveObject =
cuDeviceItem.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];
OnlineDriveFunctionInterface cuDriveFunctionInterface =
cuOnlineDriveObject.GetService<OnlineDriveFunctionInterface>();
HardwareProjection hardwareProjection = cuDriveFunctionInterface.HardwareProjection;

// To enable the setting of an Encoder, the value of p96 should
// be set to 0 (Application class == [0] Expert) if it is present
// the current G drive.
DriveParameter p96 = cuOnlineDriveObject.Parameters.Find("p96");
if (p96 != null)
{
  p96.Value = 0;
}

// Setting Encoder 1 on the drive.
// In case of encoders we have to set several enums to define an
// encoder type. These enums are: EncoderInterface, EncoderType,
// AbsoluteIncrementalFlag, and RotaryLinearFlag.

// There can be a problem, if the given enum combination is not valid.
// In that case, it has to give back a feedback.

hardwareProjection.SetEncoder(
  EncoderInterface.Terminal,
  EncoderType.HTLTTL,
  AbsoluteIncrementalFlag.Incremental,
  RotaryLinearFlag.Rotary,
  1);

// It is possible to set 2 encoders to a motor, one with encoderNumber == 1
// and the other with encoderNumber == 2
```

The following example shows how you can read out the actual encoder configuration from the drive.

Reading out the encoder configuration

```
DeviceItem cuDeviceItem = m_Device.DeviceItems[1];
DriveObject cuDriveObject =
cuDeviceItem.GetService<DriveObjectContainer>().DriveObjects[0];
DriveFunctionInterface cuDriveFunctionInterface =
cuDriveObject.GetService<DriveFunctionInterface>();

HardwareProjection encoderProjection = cuDriveFunctionInterface.HardwareProjection;

// Before setting an encoder, p96 should be 0
// (See section "Projecting EncoderConfiguration")

encoderProjection.SetEncoder(
  EncoderInterface.Terminal,
  EncoderType.HTLTTL,
  AbsoluteIncrementalFlag.Incremental,
  RotaryLinearFlag.Rotary,
  1);

EncoderConfiguration encoderConfiguration =
encoderProjection.GetCurrentEncoderConfiguration(1);
```

The following example shows how you can configure the actual encoder configuration offline.

Configuring an encoder offline

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;
```

Configuring an encoder offline

```
// Project encoder configuration in Offline state
DeviceItem cuDeviceItem = m_Device.DeviceItems[1];

DriveObject cuDriveObject =
cuDeviceItem.GetService<DriveObjectContainer>().DriveObjects[0];
DriveFunctionInterface cuDriveFunctionInterface =
cuDriveObject.GetService<DriveFunctionInterface>();
HardwareProjection hardwareProjection = cuDriveFunctionInterface.HardwareProjection;

// Before setting an encoder type, p96 should be 0
// (See section "Projecting EncoderConfiguration")

// To Project an EncoderConfiguration, first set e.g. Encoder 1 with .SetEncoder(...).

hardwareProjection.SetEncoder(
  EncoderInterface.Terminal,
  EncoderType.HTLTTL,
  AbsoluteIncrementalFlag.Incremental,
  RotaryLinearFlag.Rotary,
  1);

// Get the current configuration of Encoder 1.
EncoderConfiguration config = hardwareProjection.GetCurrentEncoderConfiguration(1);

// Fill out Encoder 1's configuration values.
config.RequiredConfigurationEntries.ToList().ForEach(ce =>
{
  switch (ce.Name)
  {
    case "p405.0":
      ce.Value = 0;
      break;
    case "p405.1":
      ce.Value = 1;
      break;
    case "p408":
      ce.Value = 1024;
      break;
    case "p425":
      ce.Value = 2048;
      break;
    default:
      break;
  }
});

// Project the Encoder 1 configuration to the device.
bool result =
cuDriveFunctionInterface.HardwareProjection.ProjectEncoderConfiguration(config, 1);
```

The following example shows how you can configure the actual encoder configuration online.

Configuring an encoder online

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;
```

Configuring an encoder online

```
// Project encoder configuration in Online state
DeviceItem cuDeviceItem = m_Device.DeviceItems[1];
DeviceItem cuDeviceItemForOnline = m_Device.DeviceItems[0];
//You need the rack for going online on G120 Device, which is .DeviceItems[0]

// GoOnline...
// To use these function in Online, you have to use the OnlineDriveFunctionInterface
OnlineDriveObject onlineDriveObject =
cuDeviceItem.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];
OnlineDriveFunctionInterface onlineDfi =
onlineDriveObject.GetService<OnlineDriveFunctionInterface>();
HardwareProjection hardwareProjection = onlineDfi.HardwareProjection;

// Before setting an encoder, p96 should be 0
// (See section "Projecting EncoderConfiguration")

// To Project an EncoderConfiguration, first set e.g. Encoder 1 with .SetEncoder(...).

hardwareProjection.SetEncoder(
  EncoderInterface.Terminal,
  EncoderType.HTLTTL,
  AbsoluteIncrementalFlag.Incremental,
  RotaryLinearFlag.Rotary,
  1);

// Get the current configuration of Encoder 1.
EncoderConfiguration config = hardwareProjection.GetCurrentEncoderConfiguration(1);

// Fill out Encoder 1's configuration values.
config.RequiredConfigurationEntries.ToList().ForEach(ce =>
{
  switch (ce.Name)
  {
    case "p405.0":
      ce.Value = 0;
      break;
    case "p405.1":
      ce.Value = 1;
      break;
    case "p408":
      ce.Value = 1024;
      break;
    case "p425":
      ce.Value = 2048;
      break;
    default:
      break;
  }
});

// Project the Encoder 1 configuration to the device.
bool result = onlineDfi.HardwareProjection.ProjectEncoderConfiguration(config, 1);
```

The following example shows how you can write the configuration entry to the console.

Write out configuration entry

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
MotorConfiguration motorConfig = hardwareProjection.GetCurrentMotorConfiguration(0);

foreach (var configurationEntry in motorConfig.RequiredConfigurationEntries)
{
  Console.WriteLine(configurationEntry.Name);
}
EncoderConfiguration encConfig = hardwareProjection.GetCurrentEncoderConfiguration(1);

foreach (var configurationEntry in encConfig.RequiredConfigurationEntries)
{
  Console.WriteLine(configurationEntry.Name);
}
```

3.11 Configuring devices

The following example shows how you can configure S120 and G120 drive devices offline.

Configuring S120 and/or G120 drive devices offline

```
using Siemens.Engineering.MC.Drives
using Siemens.Engineering.MC.Drives.DFI
using Siemens.Engineering.MC.Drives.Enums;

DriveObject driveObject = ...
DriveFunctionInterface dfi = driveObject.GetService<DriveFunctionInterface>();
HardwareProjection hardwareProjection = dfi.HardwareProjection;

// dfi can be null in case of the actual driveobject does not support it.
// hardwareProjection can be null, if the actual driveObject does not support it.
// For example: On G120 drives, you have to use the CU as driveobject. On S120
// drives, you have to use the MotorModul as driveObject.
```

The following example shows how you can configure G120 drive devices online.

Configuring G120 drive devices online

```
using Siemens.Engineering.MC.Drives
using Siemens.Engineering.MC.Drives.DFI
using Siemens.Engineering.MC.Drives.Enums;

OnlineDriveObject onlineDriveObject = ...
OnlineDriveFunctionInterface onlineDfi =
onlineDriveObject.GetService<OnlineDriveFunctionInterface>();
HardwareProjection hardwareProjection = onlineDfi.HardwareProjection;

// onlineDfi can be null in case of the actual onlineDriveObject
// does not support it or if the device is offline.
// hardwareProjection can be null, if the actual onlineDriveObject
// does not support it.
// For example: On G120 drives, you have to use the CU as onlineDriveObject. On
// S120 drives, you have to use the MotorModul as onlineDriveObject.
```

3.12 Creating a component for a drive component (S120 only)

You have the option of creating a component below a drive component for the S120.

Also enter a type designation to distinguish between the various encoders. The possible type designations and restrictions for encoders are described in the table below.

The following example shows how to create a component below a drive component.

Creating a motor and an encoder below a Motor Module

```
DeviceItem subModul = sdrDevice.PlugNew(@"OrderNumber:6SL3xxx-xxxxx-xxxx",
"MotorModul", 65535);

//Plug a motor to the motor modul
subModul.Container.PlugNew(@"OrderNumber:1PH2092-4WG4x-xxxx",
"Motor_1",65535);

//Plug an encoder to the motor modul
subModul.Container.PlugNew(@"OrderNumber:XExxxxx-xxxxx-xxxx//DRIVE-
CLIQ.202", "Encoder_1",65535);
```

Type designations for encoders and restrictions

The following restrictions apply when inserting encoders via Openness:

- Only an unspecific Sensor Module can be created for some encoders when inserting via Openness. In this case, you must configure the specific type of the Sensor Module in the TIA Portal.
- Maximum two encoders can be inserted for one Motor Module.

The following table lists the available type designations for encoders.

DRIVE-CLiQ	Resolver	sin/cos	SSI	sin/cos+SSI	HTL/TTL	HTL/TTL +SSI	EnDat 2.1
DRIVE-CLiQ.202	Resolver.0	SIN_COS.0	SSI.0	SIN_COS_+_SSI.0	HTL_TTL.0	HTL_TTL +SSI.0	EnDat_2.1.2051
DRIVE-CLiQ.204	Resolver.1001	SIN_COS.2001	SSI.3081	SIN_COS_+_SSI.2081	HTL_TTL.3001	HTL_TTL +SSI.3088	EnDat_2.1.2052
DRIVE-CLiQ.212	Resolver.1002	SIN_COS.2002	SSI.3082	SIN_COS_+_SSI.2082	HTL_TTL.3002	HTL_TTL +SSI.3090	EnDat_2.1.2053
DRIVE-CLiQ.214	Resolver.1003	SIN_COS.2003	SSI.9999	SIN_COS_+_SSI.2083	HTL_TTL.3003	HTL_TTL +SSI.9999	EnDat_2.1.2054
DRIVE-CLiQ.242	Resolver.1004	SIN_COS.2004		SIN_COS_+_SSI.2084	HTL_TTL.3005		EnDat_2.1.2055
DRIVE-CLiQ.244	Resolver.9999	SIN_COS.2005		SIN_COS_+_SSI.9999	HTL_TTL.3006		EnDat_2.1.2151
DRIVE-CLiQ.9999		SIN_COS.2006			HTL_TTL.3007		EnDat_2.1.9999
DRIVE-CLiQ.10100		SIN_COS.2007			HTL_TTL.3008		EnDat_2.1.10100
		SIN_COS.2008			HTL_TTL.3009		

DRIVE-CLiQ	Resolver	sin/cos	SSI	sin/cos+SSI	HTL/TTL	HTL/TTL +SSI	EnDat 2.1
		SIN_COS.2010			HTL_TTL.3011		
		SIN_COS.2012			HTL_TTL.3020		
		SIN_COS.2013			HTL_TTL.3109		
		SIN_COS.2110			HTL_TTL.9999		
		SIN_COS.2111					
		SIN_COS.2112					
		SIN_COS.9999					

3.13 Defining the motor type and motor configuration

The following examples show how you can set a motor type for G120 drives via the device configuration:

Setting the motor type offline via the hardware configuration

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;

//Offline (Only on G120 drives)
DriveFunctionInterface dfi = ...
HardwareProjection hardwareProjection = dfi.HardwareProjection;

// hardwareProjection can be null in case of the actual driveObject
// does not support activation.
// Setting the required MotorType and DriveDataSetNumber on the drive.
// It is only supported on G120 drives.

//First parameter is the MotorType, Second parameter is the DriveDataSetNumber
hardwareProjection.SetMotorType(MotorType.InductionMotor, 0);

// There can be a problem, if the selected MotorType is not available
// on the drive. In that case, it has to give back a feedback.
```

Setting the motor type online via the hardware configuration

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;

//Online (Only on G120 drives)
OnlineDriveFunctionInterface onlineDfi = ...
HardwareProjection hardwareProjection = onlineDfi.HardwareProjection;

// hardwareProjection can be null in case of the actual
// onlineDriveObject does not support activation.
// Setting the required MotorType and DriveDataSetNumber on
// the drive. It is only supported on G120 drives.

// First parameter is the MotorType, Second parameter is the DriveDataSetNumber
hardwareProjection.SetMotorType(MotorType.InductionMotor, 0);

// There can be a problem, if the selected MotorType is not available on
// the drive. In that case, it has to give back a feedback.
```

The following examples show how you can read out the motor type for G120 drives from the drive:

Determining the motor configuration offline

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;
```

Determining the motor configuration offline

```
// Offline
// WARNING: You have to set the MotorType on G drives before
// you would like to get the current configuration, otherwise
// you will get an exception/feedback, which informs you that
// there is no motor set.
// On S120 drives, you don't have to do that, but you have to
// plug a motor to motor modul.( later on)

HardwareProjection hardwareProjection = dfi.HardwareProjection;
// Get the current motor configuration
// It needs a datasetnumber, which is currently only supported on G120 drives.
MotorConfiguration motorConfiguration = hardwareProjection.GetCurrentMotorConfiguration(0);
```

Determining the motor configuration online

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;
// Online
// WARNING: You have to set the MotorType on G drives before
// you would like to get the current configuration, otherwise
// you will get an exception/feedback, which informs you that
// there is no motor set.
// On S120 drives, it is not possible??

HardwareProjection hardwareProjection = onlineDfi.HardwareProjection;
// Get the current motor configuration
// It needs a datasetnumber, which is currently only supported on G120 drives.
MotorConfiguration motorConfiguration = hardwareProjection.GetCurrentMotorConfiguration(0);
```

The following example shows how you can create a motor configuration for G120 drives:

Configuring a motor configuration

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;
```

Configuring a motor configuration

```
DeviceItem cuDeviceItem = m_Device.DeviceItems[1];
DriveObject cuDriveObject =
cuDeviceItem.GetService<DriveObjectContainer>().DriveObjects[0];
DriveFunctionInterface cuDriveFunctionInterface =
cuDriveObject.GetService<DriveFunctionInterface>();

HardwareProjection hardwareProjection = cuDriveFunctionInterface.HardwareProjection;
hardwareProjection.SetMotorType(MotorType.InductionMotor, 0);

MotorConfiguration config = hardwareProjection.GetCurrentMotorConfiguration(0);

config.RequiredConfigurationEntries.ToList().ForEach(ce =>
{
  switch (ce.Number)
  {
    case 305:
        ce.Value = 20;
        break;
    case 307:
        ce.Value = 30;
        break;
    case 311:
        ce.Value = 200;
        break;
    case 304:
        ce.Value = 450;
        break;
    case 310:
        ce.Value = 50;
        break;
    case 335:
        ce.Value = 1;
        break;
    default:
        break;
  }
});
bool result =
cuDriveFunctionInterface.HardwareProjection.ProjectMotorConfiguration(config, 0);
```

3.14 Reading and writing parameters

The following example shows how to read and write values of drive parameters. You require a drive object for access.

Access to parameters

```
using Siemens.Engineering.MC.Drives;
```

Access to parameters

```
//Access a parameter via its name
DriveParameter parameter = driveObject.Parameters.Find("p5391[0]");

//Example of reading parameter attributes
if (parameter != null)
{
  Console.WriteLine("The Name of the parameter is : " + parameter.Name);
  Console.WriteLine("The value of the parameter is : " +
parameter.Value.ToString());
  Console.WriteLine("The minvalue of the parameter is : " +
parameter.MinValue);
  Console.WriteLine("The MaxValue of the parameter is : " +
parameter.MaxValue);
  Console.WriteLine("The Unit of the parameter is : " + parameter.Unit);

  //Example for write:
  parameter.Value = 60;
}

// Access a parameter via Number and ArrayIndex// Note that
//   - arrayless parameters (e.g. p96 on G120) are indexed by -1
//   - parameters that consist of only a bit array do not count as
//     array parameters, they are indexed by -1 as well and the
//     further bit values can be accessed by the 'Bits' property
//     of the given parameter (e.g. r2139 on G120). For further
//     information about accessing bit parameters see the next
//     section of this code snippet

DriveParameter r947_6 = driveObject.Parameters.Find(947, 6); // returns
r947[6]if (r947_6 != null)
{
  Console.WriteLine("The Name of the parameter is : " + r947_6.Name);
  Console.WriteLine("The value of the parameter is : " +
r947_6.Value.ToString());
}

//Accessing bit values
DriveParameter p2720 = driveObject.Parameters.Find(2720, 0);if (p2720 !=
null)
{
  // Note that in general, pXXX.Bits[YYY] is not necessarily equivalent to
pXXX.YYY.
  // pXXX.Bits is an array of available bit values in ascending order by
their names
  // and not an array of bit values indexed by their names.
  // For example: let the available bit values be [pXXX.0, pXXX.2, pXXX.3],
then
  // pXXX.Bits[2] == pXXX.3, not pXXX.2

  DriveParameter p2720Bit1 = p2720.Bits[1]; // returns p2720[0].1
  if (p2720Bit1 != null)
  {
    Console.WriteLine("The name of the parameter is : " + p2720Bit1.Name);
    Console.WriteLine("The value of the second bit of the parameter is : "
+ p2720Bit1.Value.ToString());
  }
```

Access to parameters

```
}

//Get the enum values of a parameter
DriveParameter r47 = driveObject.Parameters.Find("r47");
foreach (var enumItem in r47.EnumValueList)
{
  Console.WriteLine("Enum value: " + enumItem.Key.ToString() + " = " +
enumItem.Value);
}
```

3.15 Reading and writing parameters online

The following example shows how to obtain a list with the available online parameters. You require an online drive object for access.

The read and write access to individual online parameters from the parameter list is identical to that in the Reading and writing parameters (Page 56) example.

Access to online parameters

```
using Siemens.Engineering.MC.Drives;
DeviceItem item = ... //device item of the CU (e.g. :
project.Devices[0].Items[0].Items[0])
OnlineDriveObject onlineDriveObject =
item.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];
if (onlineDriveObject != null)
{
  var parameters = onlineDriveObject.Parameters;
}
```

3.16 Saving the parameterization

The following example shows how you can determine the parameterization from S120, S210 or G120 drives.

Determining the parameterization online from the drives

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;

OnlineDriveObject onlineDriveObject =
item.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];
OnlineDriveFunctionInterface onlineDfi =
onlineDriveObject.GetService<OnlineDriveFunctionInterface>();
DriveDomainFunctions driveDomainFunctions = onlineDfi.DriveDomainFunctions;

// onlineDfi can be null in case of the actual onlineDriveObject
// does not support it or if the device is offline.

// driveDomainFunctions can be null, if the actual onlineDriveObject
// does not support it.

// For example: On G120 and S210 drives, you have to use the CU to get the
onlineDriveObject.
// Please, pay attention at S120 drives. Here you can use any module (CU, MotorModul,
lineModul) as onlineDriveObject but preferable the CU, because it will run on CU!
```

The following example shows how you can copy the parameterization for S120, S210 or G120 drives from RAM to ROM - and therefore retentively save the parameter assignment.

Backing up from RAM to ROM

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;

//In case of G120 device.
DeviceItem cuDeviceItem = m_Device.DeviceItems[1];
OnlineDriveObject cuDriveObject =
cuDeviceItem.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];

//In case of S120 and S210 device you should generally get the CU driveobject.
DeviceItem cuDeviceItem = m_Device.DeviceItems[0];OnlineDriveObject cuDriveObject =
cuDeviceItem.GetService<OnlineDriveObjectContainer>().OnlineDriveObjects[0];

// To use the function, you have to use the OnlineDriveFunctionInterface.
OnlineDriveFunctionInterface cuDriveFunctionInterface =
cuDriveObject.GetService<OnlineDriveFunctionInterface>();
DriveDomainFunctions driveDomainFunctions = cuDriveFunctionInterface.DriveDomainFunctions;

// This function will perform a RAM to ROM copy on all device.
// It is important that the CU should be in online state before you call this function
anyway the program will give an exception.
// This call may take a few moments before be finished.
bool result = DriveDomainFunctions.PerformRAMtoROMCopyAllDriveObject();
```

3.17 Using Safety Integrated telegrams

The following example shows how you can use Safety Integrated telegrams (e.g. 30) in Openness.

Using Safety Integrated telegrams

```
using Siemens.Engineering.MC.Drives;
TelegramComposition telegrams = drvObj.Telegrams;

//Add safety telegram
const int tgrmNumber = 30;
drvObj.Telegrams.InsertSafetyTelegram(tgrmNumber);

//Find safety telegram
Telegram safetyTgrm = drvObj.Telegrams.Find(TelegramType.SafetyTelegram);

// Get and set safety telegram attributes
uint watchDogTime = (uint)safetyTgrm.GetAttribute("Failsafe_FMonitoringtime");

safetyTgrm.SetAttribute("Failsafe_FMonitoringtime", 300);

const int newSafetyTelegramNumber= 900;
if (safetyTgrm.CanChangeTelegram(newSafetyTelegramNumber))
{
  safetyTgrm.TelegramNumber = newSafetyTelegramNumber;
}

//Remove Safety telegram
drvObj.Telegrams.EraseTelegram(TelegramType.SafetyTelegram);
```

3.18 Inserting and extending telegrams

The following example shows how to insert an extension and change the size of a standard telegram. You require a drive object for access.

Inserting an extension and changing the size of a standard telegram

```
using Siemens.Engineering.MC.Drives;
TelegramComposition telegrams = drvObj.Telegrams;
Telegram telegram = telegrams.Find(TelegramType.MainTelegram);

Console.WriteLine("The Cu has the telegram: " + telegram.TelegramNumber);
Console.WriteLine("The Setpoint channel-specific size of the telegram is: "
+ telegram.GetOutputSize());

foreach (var address in telegram.Addresses)
{
  if (address.IoType == AddressIoType.Output)
  {
    Console.WriteLine("The Setpoint channel-specific starting address of
the connected PLC is: " + address.StartAddress);
  }
  else if(address.IoType == AddressIoType.Input)
  {
    Console.WriteLine("The Actual value channel-specific starting address
of the connected PLC is: " + address.StartAddress);
  }
}

// Add an additional Telegram
if (drvObj.Telegrams.CanInsertAdditionalTelegram(2,4))
{
  drvObj.Telegrams.InsertAdditionalTelegram(2,4);
}

// Add a 3 word extension to the main telegram
Telegram mainTelegram == drvObj.Telegrams.Find(TelegramType.MainTelegram);
Int32 newSize = mainTelegram.GetSize(AddressIoType.Input) + 3;
if (mainTelegram.CanChangeSize(AddressIoType.Input, newSize, true))
{
  mainTelegram.ChangeSize(AddressIoType.Input, newSize, true)
}
```

3.19 Using torque telegrams

The following example shows how you can use torque telegrams (e.g. 750) in Openness.

Using torque telegrams

```
using Siemens.Engineering.MC.Drives;
const int torqueTelegramNumber = 750;
TelegramComposition telegrams = drvObj.Telegrams;

// Add a new torque telegram
if (drvObj.Telegrams.CanInsertTorqueTelegram(torqueTelegramNumber))
{
  drvObj.Telegrams.InsertTorqueTelegram(torqueTelegramNumber);
}

// Find torque telegram
Telegram torqueTelegram = drvObj.Telegrams.Find(TelegramType.TorqueTelegram);
```

3.20 Restoring factory settings

The following example shows how you can restore the factory settings for S120, S210 or G120 drives.

Restoring factory settings

```
using Siemens.Engineering.MC.Drives;
using Siemens.Engineering.MC.Drives.DFI;
using Siemens.Engineering.MC.Drives.Enums;

OnlineDriveObject onlineDriveObject = ...
OnlineDriveFunctionInterface onlineDfi =
onlineDriveObject.GetService<OnlineDriveFunctionInterface>();
DriveDomainFunctions driveDomainFunctions = onlineDfi.DriveDomainFunctions;

// onlineDfi can be null in case of the actual onlineDriveObject
// does not support it or if the device is offline.
// saveParametrization can be null, if the actual onlineDriveObject
// does not support it.
// For example: On G120 and S210 drives, you have to use the CU to get the
onlineDriveObject.
// S120 drives, you can use any modul (CU, MotorModul, lineModul) as onlineDriveObject but
preferable the CU.
```