Tuner4TRONIC® DLL 4

Reference Manual

Version 4.1.0.0

2021-04-21

Light is OSRAM

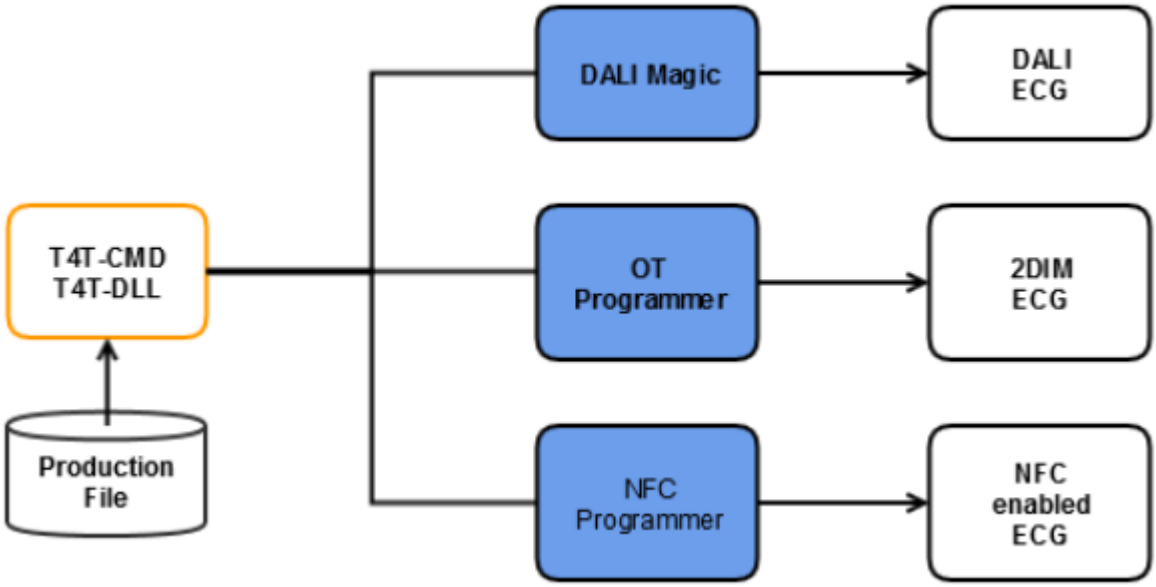
Please note:
All information in this guide has been prepared with great care. OSRAM, however, does not accept liability for possible errors, changes and/or omissions. Please check www.osram.com or contact your sales partner for an updated copy of this guide. This technical application guide is for information purposes only and aims to support you in tackling the challenges and taking full advantage of all opportunities the technology has to offer. Please note that this guide is based on own measurements, tests, specific parameters and assumptions. Individual applications may not be covered and need different handling. Responsibility and testing obligations remain with the luminaire manufacturer/OEM/application planner.

Table of Content

Introduction

The T4T-DLL performs the programming of OSRAM ECGs based on production files created by the T4T-D software. Therefore the main class needs to be passed a reference to the production file. At the start of programming, the T4T-DLL connects to the required PI; if no PI is available, not the right PI type is available or too many PIs are connected to the PC, an error message will be returned. After programming, statistic data of the programmed ECG(s) in the luminaire configuration can be inquired. The PI connection is released when a new production file requiring a different PI type is opened or the class is discarded.

The following figure gives a high-level overview of the required environment, shows the components that are involved and how they are connected.



Scope

This document describes the API of the T4T-DLL (Tuner4TRONIC) which allows programming of OSRAM ECGs based on production files created with the T4T-D application. The error codes and when they might occur is described. Moreover examples are given how to integrate this API into third party software (limited to .NET applications).

Acronyms and Abbreviations

PI	Programming Interface
DALI Magic	DALI Magic OSRAM's USB to DALI converter, PI for DALI ECGs
OT Programmer	OSRAM's PI for 2DIM devices
T4T	Tuner4TRONIC Software Suite
T4T-D	T4T Development Software
API	Application Programming Interface

Dependencies

- In order to work with the T4T-DLL files provided in the ZIP archive need to be extracted into the same folder
- The .Net Framework 4.6 must be installed on the PC running the application using the T4T-DLL
- Only one Programming Interface, either a "DALI magic", an "OT Programmer", or a NFC Programmer, depending on which ECG type shall be programmed, is connected to the PC using the T4T-DLL.
- To ensure a proper function of the T4T-DLL together with DALI magic, the firmware of the DALI Magic must be at least version 2.50. Otherwise an error will be returned. OSRAM provides separate tool to update the DALI Magic FW as part of the T4T Suite.

Important Notes

If the programming of a factory-new device was not confirmed by "Success" (0) it must not be assumed that the parameters of the ECG still contain the default values. As the programming may have been interrupted, some values may have been changed.

Changes in the API

There are some changes in the T4T-DLL in order to support new features of T4T. The API changes in version 4.0.0 are listed below:

- Properties: The properties "Config_ID" and "isDirectProgrammingEnabled" are deprecated and were removed from this documentation.
- Events: The event "ReadParameterStatus" is deprecated and was removed from this documentation
- Methods: The methods "EnableLabelPrinting", "SetSpoolingFolderPath", "SetLabelDefinitionFile" and "QueryParameter" are deprecated and were removed from this documentation.

API Specification

This section defines the class that is exposed in the T4T-DLL, namely the **ProgrammingAPI**. The main DLL file is **ProgrammingAPI.dll**. Refer to that file in the application that shall work with the T4T-DLL.

The ProgrammingAPI Class

This class is responsible for programming of the luminaire based on the production file (*.osrtup) that is passed as parameter. This class validates the file that is passed, de-serializes the file to the internal objects and connects to the Programming Interface. Then programming of connected devices can be started. The ProgramingManager raises the events **ProgrammingStatusChanged** and **ProgressChanged** which informs about the programming status and the progress after programming was started.

Properties

Version

This property returns the T4T-DLL version. The format is (Major.Minor.Revision), for example, the result could be "4.0.0".

Luminaire_Name

This property returns the luminaire name that is stored in the currently loaded production file. Returns an empty string if no or unusable production file is loaded.

Luminaire_OrderCode

This property returns the luminaire order code that is stored in the currently loaded production file. Returns an empty string if no or an unusable production file is loaded.

Luminaire_Image

This property returns the path to the decoded luminaire image that is included in the currently loaded production file. Returns an empty string if no or an unusable production file is loaded or the production file has no luminaire image included.

Luminaire_Description

This property returns luminaire description text that is stored in the currently loaded production file. Returns an empty string if no or unusable production file is loaded.

Luminaire_CustomerProject

This property returns "customer/project" text that is stored in the currently loaded production file. Returns an empty string if no or unusable production file is loaded.

Luminaire_Configuration

This property returns a list of structs for the ECGs that are included the currently loaded luminaire configuration. The struct includes:

- the ECG type name
- the GTIN of the ECG type if available for the ECG type or an empty string
- the Basic Code if available for the ECG type or an empty string
- the NAED Code if available for the ECG type or an empty string
- the multiplicity attribute for this ECG type in the luminaire configuration. 0 stands for an undefined number, while 1…64 represent a predefined number.

In case no production file is loaded, an empty list will be returned.

Events

ProgrammingStatusChanged

This event is triggered when there is change in the programming status or of the Programming Interface status change after initiated programming. All possible (interim) programming states are listed in Table 1. If an error occurs, the detailed error code is returned via the WriteCompleteCallback.

ProgressChanged

This event is triggered when there is change the programming progress. The returned value represents the completed percentage between 1 and 100%.

DeviceInfoReceived

This event is triggered when ECG information like Serial Numbers of the connected ECGs are available. The event passes a list of structures which contain information about the programmed ECGs. The structure includes:

- GTIN
- Serial Number
- FW major
- HW major

This structure may be enhanced later.

WriteParameterStatus

This event is triggered when the WriteParameter function is completed. This event contains the ReturnCode indicating the status of the operation and the written value (as requested of modified/clipped).

Methods

LoadProductionFile

This method takes the production file path and name and loads the production file (as generated by T4T-D). Additionally a connection to the required Programming Interface type is established. When the call successfully returns, the production file is load and an appropriate PI is connected. The program method cannot be executed when this call was not successful.

Parameters

1. **File Name** (string): production file name with osrtup extension including the full absolute path.

Returns

ReturnCode (enum): Code that described the status of reading the production file and connecting the PI. The potential general, production file or programming interface related messages are listed in Table 1.

LoadProductionFile

This method takes the production file path, the PI_Type and ReaderID and loads the production file. Additionally, it establishes a connection to the selected PI_Type based on the ReaderID. When the call successfully returns, the production file is loaded and the selected PI is connected. The program method cannot be executed when this call was not successful.

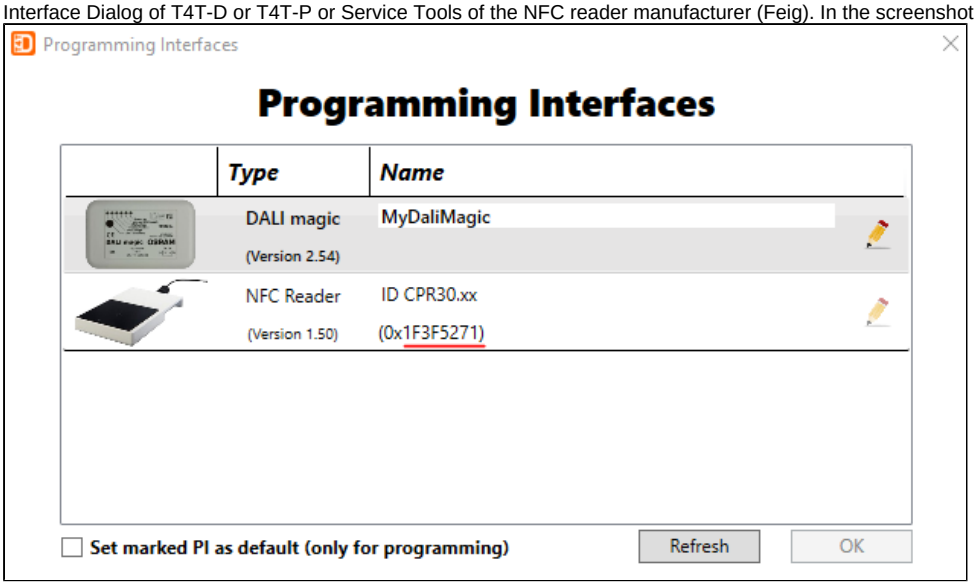
Parameters

1. **File Name** (string): The name of the production file with osrtup extension including the full, absolute path.

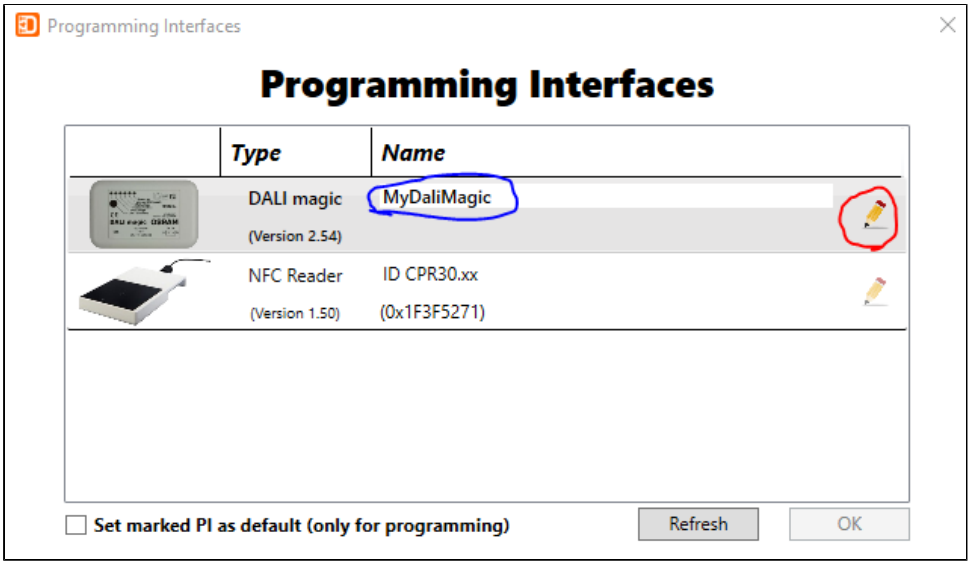
2. **PI_Type** (enum): The PI Type can be **NFC**, **DALI** or **OSRSER**; the enum for PI_Type are listed in Table 3. If PI_Type parameter is "**auto**", then the DLL will load the file and establish a connection to the required Programming Interface type. When multiple matching PIs are present when running the DLL, an error will be thrown.

3. **PI_Name** (string): The name of the PI to be used by the ProgrammingManager. The PI name is used to differentiate between multiple PIs when multiple PIs of the same Type are connected to the PC running the DLL. The required name depends on the PI_Type:

- For PI_Type "***NFC***", the PI_Name is the ReaderID in hexformat. The ReaderID of your device can be viewed in the Programming Interface Dialog of T4T-D or T4T-P or Service Tools of the NFC reader manufacturer (Feig). In the screenshot marked in red).



- For PI_Type "***DALI***", PI_Name is the name of the DALI Magic, which can be viewed and changed in the Programming Interface Dialog of T4T-D or T4T-P. In the screenshot first click the pencil button (marked red) to rename the device so you can use this name.



- For PI_Type "***OSRSER***", the COM Port number is used as PI_Name. The COM Port number can be viewed and changed in the Windows Device Manager.

Example
A correct implementation would look like this:

T4T-CMD.exe T4T -PI “NFC:1DBB5464“ myproductionfile.osrtup

or

T4T-CMD.exe T4T -PI NFC:1DBB5464 myproductionfile.osrtup

Note: The “ ” are only needed if the PI name has a space in its name.

The PI_Name may be left empty, then the DLL takes the available interface of that PI Type. In case of no or multiple interfaces of that PI_Type are connected, there will be an error message.

Returns

ReturnCode (enum): A Code that described the status of reading the production file and connecting the PI. The related error messages are listed in Table 1.

Program

This method programs the data of the loaded production file. It uses the programming settings (Reworks/First Programming, OEMCode, and Verify) as stored in the currently loaded production file.
If a previous shortcut situation at the PI was detected, the power will be switched on again.

Parameters

1. **WriteCompleteCallback**: User defined callback method that will be triggered once the programming is completed. The signature of call back method must be:

    void WriteCompleteCallback (ReturnCode programmingStatus)

Returns

1. The **ReturnCode** is either "Success" or one of the programming related messages as listed in Table "Result Codes".

Program

This method programs the data of the loaded production file. It enables to overwrite the programming settings that are stored in currently loaded production file based on the parameters that are passed, if permitted by the production file.
If a previous shortcut situation at the PI was detected, the power will be switched on again.

Parameters

1. **OEMCode** (int): Four digits unlock code for programming of the protected features.

2. **VerifyProgramming** (bool): Indicates if programming shall be followed by verification (true) or not (false). Note that this is slows down the programming process.

3. **EnableSingleProgramming** (bool): True indicates that only a single drivers is connedected, hence no addressing will be performed**.**

4. **DisableFamilyProgramming** (bool): True indicates that no family programming shall be executed, i.e. programming will only be performed in case of an exact match of production file and connected ECG. False indicates that also an ECG of the same or later family will be accepted for programming.

5. WriteCompleteCallback (delegate): User defined callback method that will be triggered once the programming is completed. The signature of call back method must be:

    void WriteCompleteCallback (ReturnCode programmingStatus)

Returns

The **ReturnCode** is either "Success" or one of the programming related messages as listed in Table "Result Codes".

Program

This method programs the data of the loaded production file. It enables to overwrite the programming settings that are stored in currently loaded production file based on the parameters that are passed, if permitted by the production file. If a previous shortcut situation at the PI was detected, the power will be switched on again.

Parameters

1. **MasterKey** (string): The master unlock code for programming of the protected features in the hexadecimal format 0x00000000. The length of the unlock code must be 4 bytes.

2. **VerifyProgramming** (bool): Indicates if programming shall be followed by verification (true) or not (false). Note that this is slows down the programming process.

3. **EnableSingleProgramming** (bool): True indicates that only a single drivers is connedected, hence no addressing will be performed**.**

4. **DisableFamilyProgramming** (bool): True indicates that no family programming shall be executed, i.e. programming will only be performed in case of an exact match of production file and connected ECG. False indicates that also an ECG of the same or later family will be accepted for programming.

5. **WriteCompleteCallback** (delegate): User defined callback method that will be triggered once the programming is completed. The signature of call back method must be:

    void WriteCompleteCallback (ReturnCode programmingStatus)

Returns

The **ReturnCode** is either "Success" or one of the programming related messages as listed in Table "Result Codes".

Abort

Abort the programming. If programming is in progress, the WriteCompleteCallback returns the code "ProgrammingAborted".

QueryDeviceInfo

This method sends a request for device information like GTIN, Serial Number etc. Once available, a DeviceInfoReceived event will be triggered with these information of the present luminaire configuration.

EnableLabelPrinting

This method enables label printing.

Parameters

1. Parameter type(bool): Indicates whether label printing is enabled or not.

SetSpoolingFolderPath

This method sets the path into which the csv file for label printing will be created, if the label printing feature is enabled. On successful programming of the ECG(s), a csv file with all the parameter data programmed to the ECG(s) is generated.

Parameters

1. **Path** (string): Indicates the path to create the csv file.

SetLabelDefinitionFile

This method sets the label definition file path. This field is optional.

Parameters

1. **Path** (string): Indicates the path of the label definition file.

WriteParameter

This method modifies the value of the specified parameter on a connected ECG. On successfully programming, the value a WriteParameterStatus event will be triggered. That returns the status if value is programmed successfully, the value is clipped or if operation failed for any reason.

Note: This function does not support luminaire configurations with more than one ECG.

Parameters

1. ParameterType (enum): Indicates the parameter type that has to be modified.

2. SetValue (int): Indicates the actual value to be programmed to ECG.

WriteParameter

This method modifies the value of the specified parameter on a connected ECG. On successfully programming, the value a WriteParameterStatus event will be triggered. That returns the status if value is programmed successfully, the value is clipped or if operation failed for any reason.

Note: This function does not support luminaire configurations with more than one ECG.

Parameters

1. ParameterType (enum): Indicates the parameter type that has to be modified.

2. SetValue (hexstring): Indicates the actual value to be programmed to ECG, in particular useful for Luminaire Data - Vendor Specific Content (6).

Note: For DLL version 4.0.0 and higher only these parameter types are supported:

- Luminaire GTIN (ApplicationConstants.ParameterType.GTIN)
- Luminaire Identification Number (ApplicationConstants.ParameterType.IdentificationNumber)
- Luminaire Content Format Version (ApplicationConstants.ParameterType.ContentFormatVersion)
- Luminaire Vendor Specific Content (ApplicationConstants.ParameterType.VendorSpecificContent)

Enums

The following table summarizes the Result Codes and the Programming Status used by the T4T-DLL.

Warning: Some Return Codes have been changed compared to previous version!

Table 1: Result and Status Codes

Enum	Value	Description
General Messages		
GeneralApplicationError	1	An unspecified application error has occurred.
TrialPeriodExpired	2	Indicates that the trial version of the DLL has expired. No further usage of this DLL version is possible.
Production File related Messages		
Success	0	Production File sucessfully loaded, no (file) error discovered.
ParamFile_NotReadable	101	The Luminaire Production File doesn't exist at given path or the file cannot be opened.
ParamFile_Invalid	102	Invalid Luminaire Production File. This can be a wrong extension, wrong version (too old or too new), or no production file at all.
ParamFile_OldVersion	103	A production file of an old T4T version was provided. Use T4T-D to convert it to the current version.
ParamFile_IsReadonly	104	The production file is read only.
ParamFile_UpgradeFailed	105	An old prodcution file was passed and an error duing the automatic upgrade happend, e.g. the file version was too old to upgrade.
Programming Interface related Messages		
Success	0	Connection to appropriate PI established.
PI_NoneFound	200	No Programming Interface found or specified Programming Interface not found.
PI_FWVersionUnsupported	201	The FW version of the Programming Interface is not supported (e.g. DALI Magic with FW 2.18).
PI_WrongTypeFound	202	The found PI does not match the Luminaire configuration.
PI_TooManyFound	203	There is more than one PI of the required type connected. Hence the T4T cannot decide which one to use.
PI_Overloaded	204	Too many ECG connected to Dali Magic or a DALI shortcut was detected.
PI_CommunicationError	205	A communication error with the PI occurred. Maybe the PI was disconnected to the USB port.
Programming related Messages		
Success	0	Device(s) were programmed successfully.
NoECGConnected	300	No response was received. Either No ECG is connected or ECGs are not powered.
MoreThanOneECGConnected	301	Single ECG configuration was selected but there is more than one ECG connected.
TooManyECGsConnected	302	More ECGs than allowed/required for the luminaire configuration are connected.
WrongECGTypeOrVersionConnected	303	One or more ECGs don't match the type or FW/HW version included in the Production File. Check the IC/NAED code of the connected ECGs.
ECGProtected	304	Or more ECG is write-protected with an OEM Code but no OEM code was provided in the production file.
InvalidPassword	305	The ECG is locked with a different OEM Code than provided in the production file.

InvalidPasswordTimelocked	306	The ECG was locked with different password and due to using a wrong password is now time-locked. After first such event external SW needs to wait 5s before next programming attempt can be made.
InconsistentPasswords	307	The system level or user level password in the device do not match the provided unlock password.
ECGCommunicationFailure	308	The ECG provided no or an unexpected response.
DaliShotcutOccured	309	Programming was terminated due to a DALI shortcut.
ECGConnectionWrong	310	The connection to the ECG has wrong polarity. This is applicable only for non-DALI devices.
VerificationFailed	311	Verification of programming failed.
ProgrammingAborted	312	The programming was aborted by method "abort".
LuminaireConfigurationNotMatching	313	The Luminaire configuration is not matching with the connected configuration.
NFCECGPowerIsON	314	Programming failed because ECG is powered on.
ConnectedLuminaireCountExceedsBatchSize	315	The number of connected luminaires to be programmed exceeds the batch size mentioned in the workflow file
ECGIsProgrammedAlready	316	In case of NFC Programming ECGs cannot be programmed multiple times when a password is set. Before reporgamming it has to be powerered on in order to transport the parameters from the NFC into the FW.
MapLuminaireConfigurationFailed	317	If family programming was selected this indicates that the parameters cannot be mapped to the connected ECG because the ECG is not in the same or later family or the connected ECG only support a small range for a parameter (e.g. 1000 mA cannot be set in a device only supporting 500-700 mA)
ProgrammingFailedDueToMisMatchInRFPassword	318	In case of NFC Programming, a problem with the NFC chip occured.
NoMatchingDDFilesFoundWithGTIN	319	In case family programming was used, no matching information for the connected driver was found.
NoMatchingDDFilesFoundWithFWVersion	320	In case family programming was used, no matching information for the connected driver was found.
NoMatchingDDFilesFoundWithHWVersion	321	In case family programming was used, no matching information for the connected driver was found.
NoMachtingDDFilesFoundWithFWHWVersion	322	In case family programming was used, no matching information for the connected driver was found.
ErrorInDeviceDataBase	323	
IllegalHexStringValue	324	The received parameter hexstring contains illegal characters. Note that a hexsring may only contain {0..9a..fA..F}
ECGNfcNotSynchronized	325	In case this error is returned, please powercycle the ECG and retry programming.
Single Parameter Programming related Messages		
NotSupportedForMultiPF	400	This function call is not supported when the loaded production file contains more than one ECG.
ParameterNotSupported	401	The selected parameter is not supported by the ECG.
ParameterClipped	402	The value that was used to program the parameter was out of range and clipped to the allowed range.
Status Messages		
DeviceDetectionStarted	1000	The programming procedure was started and the first step (device detection procedure) will be started.

DALIAdressingStarted	1001	The DALI Addressing Procedure was started.
PreprocessingStarted	1002	The pre-processing was started (if applicable)
ProgrammingStarted	1003	The ECG configuration was checked found ok and programming will be started.
PostProcessingStarted	1004	The post-processing was started (if applicable)
VerificationStarted	1005	The programming was completed and verification will be started.
ModeSettingStarted	1006	The programming and verification was completed successfully and Mode setting will be started
ProgrammingFailed	1007	The programming procedure was terminated unsuccessful.
ProgrammingBatchSizeReached	1008	The number of ECGs programmed with the configuration file has reached the maximum value as defined in the ECG configuration file.
Success	0	Devices were programmed (incl. verification if applicable) and were set into new mode (if applicable). The transaction was completed successfully.

Table 2: Parameters and Parameter Type Codes

Enum	Value	Description
Parameters the can be queried or programmed with Query/Write Parameter Methods		
OperatingCurrent	0	Indicates the Operating current parameter.
TuningFactor	1	Indicates the Tuning Factor parameter.
Luminaire GTIN	3	
Luminaire Identification Number	4	
Content Format Version	5	
Vendor Specific Content	6	
Parameter Types that can be queries from a loaded production file or a connected device		
Maximum	0	Indicates that the Maximum value of the specified parameter shall be returned.
Minimum	1	Indicates that the Minimum value of the specified parameter shall be returned.
Actual	2	Indicates that actual value from the connected device shall be queried.

Table 3: PI Types

Enum	Value	Description
Auto	0	Indicates no PI type pre-selected
DALI	1	Indicates the DALI type.
OSRSER	2	Indicates the OSR-SER type.
NFC	4	Indicates the NFC type.

C# Example

Below is an example for a command line application using the T4T-DLL.

```
using System;
using System.Linq;
using ProgrammingAPI;

namespace T4T_cmd
{
    class Program
    {
        static AutoResetEvent programmingCompleteEvent = new AutoResetEvent(false);
        static void Main(string[] args)
        {
            if (args.Length > 1)
            {
                string command = args[0];
                string fileName = string.Empty;
                if (args != null && args.Count() > 1)
                    fileName = args[1];

                if (command.ToUpper() == Commands.StartProgramming)
                {
                    WriteToDevice(fileName);
                }
                else
                {
                    Console.WriteLine("Invalid Command");
                }
            }
            else
            {
                Console.WriteLine("No Parameter Received!!!");
            }
        }
        private static void WriteToDevice(string filename)
        {

            ProgrammingAPI.ProgrammingAPI.LoadProductionFile(filename);
            System.Threading.Thread.Sleep(100);

            // Unsubscribe first
            ProgrammingAPI.ProgrammingAPI.ProgressChanged -= programmingManager_ProgressChanged;
            ProgrammingAPI.ProgrammingAPI.ProgrammingStatusChanged -=
programmingManager_ProgrammingStatusChanged;
            // Subscribe to notifications
            ProgrammingAPI.ProgrammingAPI.ProgressChanged += programmingManager_ProgressChanged;
            ProgrammingAPI.ProgrammingAPI.ProgrammingStatusChanged +=
programmingManager_ProgrammingStatusChanged;

            programmingCompleteEvent.Reset();
            System.Threading.Tasks.Task.Factory.StartNew(() =>
            {
                ProgrammingAPI.ProgrammingAPI.Program(WriteCompleted);
            });

            // Waits till the thread finishes
            bool eventStatusSignalled = programmingCompleteEvent.WaitOne(20000);
        }

        /// Call back method to be triggered on programming completion
        private static void WriteCompleted(Osram.Tuner4TRONIC.Common.ApplicationConstants.ReturnCode
programmingStatus)
        {
```

```
            if (programmingStatus == Osram.Tuner4TRONIC.Common.ApplicationConstants.ReturnCode.Success)
            {
                int Code = (int)Osram.Tuner4TRONIC.Common.ApplicationConstants.ReturnCode.Success;
                Console.WriteLine("### {0} --- code: {1}", programmingStatus, Code);
            }
            else
            {
                int Code = (int)Osram.Tuner4TRONIC.Common.ApplicationConstants.ReturnCode.Success;
                Console.WriteLine("### ERROR --- message: {0} --- code: {1}", programmingStatus, Code);
            }
            programmingCompleteEvent.Set();
        }

        static void programmingManager_ProgressChanged(object sender, ValueEventArgs e)
        {
            Console.WriteLine("### Progress:" + e.ToString());
            Console.ResetColor();
        }

        static void programmingManager_ProgrammingStatusChanged(object sender, ProgrammingStatusEventArgs e)
        {
            Console.WriteLine("### Programming Status: " + e.ReturnCode.ToString());
            Console.ResetColor();
        }
    }
    public class Commands
    {
        public const string StartProgramming = "T4T";
        public const string Version = "-VERSION";
        public const string VerboseFullTag = "-VERBOSE";
        public const string VerboseShortTag = "-V";
    }
}
```