

# Проект Тип оборудования LineDisplay

# Оглавление

<b>1. ОСНОВНЫЕ ПОНЯТИЯ И ТЕРМИНЫ .....</b>	<b>4</b>
<b>2. ОПИСАНИЕ СТАНДАРТА .....</b>	<b>5</b>
2.1. Наименование типа и русскоязычный синоним .....	5
2.2. Краткое описание функциональности типа .....	5
<b>3. АЛГОРИТМЫ .....</b>	<b>5</b>
3.1. Общие алгоритмы .....	5
Создание нового экземпляра оборудования .....	5
Использование экземпляра оборудования .....	6
Удаление экземпляра оборудования .....	6
3.2. Специфические алгоритмы .....	6
<b>4. НАСТРОЙКИ .....</b>	<b>7</b>
4.1. Общие настройки .....	7
DefaultTimeOut .....	7
EventLogEnabled .....	7
4.2. Специфические настройки .....	7
Rows .....	7
Columns .....	7
<b>5. МЕТОДЫ .....</b>	<b>8</b>
5.1. Служебные методы .....	8
Init .....	8
Open .....	8
Close .....	8
GetDeviceInfo .....	9
5.2. Общие методы .....	10
GetSettings .....	10
SetSettings .....	11
ShowSettingsDlg .....	11
CheckHealth .....	12
Enable .....	12
Disable .....	13
5.3. Специфические методы .....	13
CreateWindow .....	13
DestroyWindow .....	14
DisplayLogo .....	14
DisplayText .....	14
ClearText .....	15
<b>6. ФУНКЦИИ ОБРАТНОГО ВЫЗОВА .....</b>	<b>15</b>
GetAppProperty .....	15
ErrorEvent .....	16



# 1. Основные понятия и термины

Термин	Описание
Внешнее оборудование	Любое торговое или промышленное оборудование, использующееся клиентским приложением
Тип оборудования	Регламентированное стандартом множество, объединяющее устройства с одинаковым функциональным назначением
Драйвер оборудования, Драйвер	Компьютерная программа, с помощью которой система управления внешним оборудованием получает доступ к устройству стандартным образом, с помощью интерфейса соответствующего типа оборудования, описанного в стандарте
Модель, Модель оборудования	Множество устройств одного типа оборудования, одинаковых или близких по своим характеристикам, управляемых драйвером устройства. Модель оборудования определяется драйвером устройства. В случае, если один драйвер устройства позволяет единообразно управлять одной из нескольких сходных моделей физических устройств, для системы все эти устройства являются одной и той же моделью. Напротив, если для одного и того же физического устройства существует несколько разных драйверов, для системы они являются разными моделями.
Экземпляр оборудования	Совокупность файлов и данных о физическом устройстве и драйвере (идентификатор, наименование, список настроек и т. п.), хранимых и используемых для работы с данным устройством системой управления внешним оборудованием и приложением-клиентом.
Клиентское приложение	Любое приложение, использующее для управления внешним оборудованием описанные в данном стандарте интерфейсы и алгоритмы взаимодействия с драйверами устройств.
Менеджер оборудования	Часть клиентского приложения, в которой сосредоточены сервисные процедуры и регламентированные данным стандартом интерфейсы взаимодействия с оборудованием.
Настройки устройства	Набор именованных значений, простых типов данных: String, Number, DateTime, Boolean, хранящийся вне клиентского приложения в файловом хранилище настроек. Для каждого определенного в системе устройства имеется свой набор настроек. Настройки устройства определяют параметры и режим функционирования устройства. Настройки доступны клиентскому приложению для чтения и записи с помощью специальных методов.
Уникальный идентификатор	В этом проекте – текстовое представление GUID. Идентификатор содержит 36 символов в верхнем регистре, не заключенных в фигурные скобки. Пример: 8A8910EC-78F5-41A5-9E18-7C28992CF580
Публикуемый метод	Один из определенных в описании интерфейса общих методов, либо методов типа. Все публикуемые методы имеют одинаковую структуру параметров. Входные и возвращаемые методом данные помещаются в два одномерных массива SafeArray
Функции обратного вызова	Регламентируемые данным стандартом функции клиентского приложения, доступные для вызова через интерфейс IDispatch драйвером оборудования. Функции обратного вызова предназначены для информирования приложения о событиях или передачи данных.
Событие	Событие возникает как результат изменения состояния устройства. Событие может быть вызвано внешним воздействием, либо внутренними процессами в оборудовании. Для передачи событий приложению драйвер устройства использует функции обратного вызова клиентского приложения
Объект-абстракт	Объект драйвера устройства, не ассоциированный с экземпляром оборудования. Любой объект драйвера является объектом-абстрактом до выполнения метода Open. У объекта-абстракта допустим вызов только тех методов, которые не требуют взаимодействия с физическим устройством. Объект-абстракт не может использовать функции обратного вызова, поскольку не имеет собственного идентификатора в системе.
Объект драйвера	Объект драйвера устройства, созданный клиентским приложением. Для каждого физического устройства создается отдельный экземпляр объекта драйвера
Табло покупателя,	Представляет собой обычно устройство с вакуумно-флуоресцентным,

Термин	Описание
Дисплей покупателя	светодиодным или жидкокристаллическим экраном, предназначенное для индикации цен, наименований товаров, курсов валют, также объявлений, сообщений и другой текстовой информации.
VFD	Vacuum Fluorescent Display – Вакуумный флуоресцентный дисплей
Окно	Условная прямоугольная область экрана дисплея. Все команды вывода и очистки текста работают с окнами.

## 2. Описание стандарта

### 2.1. Наименование типа и русскоязычный синоним

Наименование типа англ.:	LineDisplay
Наименование типа рус:	ДисплейПокупателя
Синоним англ.:	Line Display
Синоним рус:	Дисплей покупателя

### 2.2. Краткое описание функциональности типа

Дисплей покупателя обычно представляет собой вакуумно-флуоресцентный дисплей (VFD), имеющий две строки символов по 20 в каждой. Драйвер дисплея может выводить произвольный текст на экран устройства. Текст может выводиться в нескольких режимах (с использованием нескольких эффектов) – статично, с прокруткой или миганием. Помимо этого, драйвер позволяет управлять выводом аппаратной заставки на экран. Текст (или изображение) и режим вывода заставки предварительно записываются в энергонезависимую память дисплея с помощью специализированного ПО или средствами обработчика.

## 3. Алгоритмы

### 3.1. Общие алгоритмы

В этой главе описаны общие для всех типов оборудования алгоритмы взаимодействия приложения с драйвером устройства.

#### Создание нового экземпляра оборудования

- Приложение создает объект драйвера устройства
- Следующим вызванным методом должен быть [\\_Init](#), инициализирующий драйвер. При этом в параметрах метода драйверу передается ссылка на интерфейс IDispatch клиентского приложения. Драйвер запоминает ссылку на интерфейс для обращения к функциям обратного вызова
- У созданного объекта вызывается метод [GetDeviceInfo](#) для получения массива [DeviceInfo](#), содержащего константную информацию о драйвере (закладывается разработчиком при реализации драйвера)
- Для получения значений настроек по умолчанию, у драйвера вызывается метод [GetSettings](#).
- Если драйвер имеет собственную форму для настройки оборудования (элемент `IsSettingsDlgExist` из структуры `DeviceInfo`), у него вызывается метод `ShowSettingsDlg` для её отображения. Если форма настроек отсутствует, приложение отображает свою универсальную форму, заполнив её значениями настроек по умолчанию
- Пользователь редактирует значения в форме настроек устройства и закрывает её
- Если пользователь отказался от сохранения настроек, нажав на форме кнопку «Отмена», считается, что он отказался от создания устройства. У драйвера вызывается метод `Close` и объект драйвера уничтожается. Процедура создания устройства прерывается.
- Если использовалась форма настройки приложения, то набор настроек передается в драйвер через метод `SetSettings` для проверки корректности их заполнения (валидации)
- Приложение генерирует уникальный идентификатор для нового экземпляра оборудования и создает соответствующий ему индивидуальный каталог в хранилище настроек
- У драйвера запрашивается набор его текущих (проверенных) настроек с помощью метода `GetSetnigns`. Набор настроек данного экземпляра оборудования сохраняется в

индивидуальном каталоге хранилища настроек в файле Settings.xml (см. документ «Установка и обновление системы управления внешним оборудованием»)

- У данного объекта драйвера вызывается метод Close, после чего он выгружается из памяти приложения
- Клиентское приложение сохраняет идентификатор созданного устройства для последующего его использования (см. Использование экземпляра оборудования)

## Использование экземпляра оборудования

В этом разделе описывается алгоритм работы с ранее созданным экземпляром оборудования.

- Приложение создает объект драйвера устройства. У созданного объекта вызывается метод [Init](#), инициализирующий драйвер. При этом в параметрах метода драйверу передается ссылка на интерфейс IDispatch клиентского приложения (используется драйвером для обращения к [функциям обратного вызова](#))
- Приложение вызывает метод драйвера [SetSettings](#), передавая в параметрах метода список значений настроек устройства. Драйвер применяет новые настройки.
- Приложение вызывает метод драйвера [Open](#), в параметрах которого передается идентификатор устройства. Данный идентификатор драйвер при обращениях к [функциям обратного вызова](#)
- Начальным состоянием любого устройства после выполнения метода [Open](#), является «Выключено». В этом состоянии допускается вызов следующих методов устройства: [GetDeviceInfo](#), [GetSettings](#), [SetSettings](#), [ShowSettingsDlg](#), [CheckHealth](#), [Enable](#), [Disable](#), [Close](#). Все остальные методы будут доступны после включения устройства.
- Перед началом реального использования оборудования приложение вызывает метод [Enable](#). При выполнении метода Enable драйвер должен выполнить все необходимые операции по подготовке к работе: подгрузить необходимые ему внешние библиотеки, открыть, захватить, либо заблокировать необходимые для работы ресурсы: коммуникационные порты, файлы и т.п. Если метод [Enable](#) завершается успешно, то устройство переходит в состояние «Включено».
- У оборудования в состоянии «Включено» клиентское приложение последовательно вызывает необходимые ему публикуемые методы драйвера устройства.
- Драйвер устройства может порождать различные события посредством вызова у приложения функций обратного вызова.
- По окончании работы с драйвером устройства, приложение должно отключить его, вызвав метод [Disable](#). При исполнении данного метода драйвер должен привести физическое устройство в исходное состояние, высвободить все захваченные им в процессе работы ресурсы (закрыть открытые им файлы, коммуникационные порты), выгрузить использовавшиеся сторонние библиотеки и т.п.
- Перед уничтожением объекта драйвера устройства, приложение вызывает метод-деструктор [Close](#). Драйвер очищает все свои служебные структуры данных, в том числе очищает ссылку на интерфейс IDispatch клиентского приложения, переданную ему при вызове метода [Init](#).

## Удаление экземпляра оборудования

Для удаления экземпляра внешнего оборудования из системы, достаточно удалить индивидуальный каталог экземпляра оборудования из хранилища настроек (см. описание в документе «Программа установки системы управления внешним оборудованием.doc») и очистить в настройках клиентских приложений, использовавших данный экземпляр, все ссылки на его идентификатор.

## 3.2. Специфические алгоритмы

Драйвер дисплея покупателя может поддерживать или не поддерживать работу с окнами. Перед выводом текста на дисплей, приложение должно создать окно методом CreateWindow, в которое будет выводиться данный текст. По умолчанию, автоматически создается окно с номером «0», адресующее всю область экрана, оно же и будет единственным, если дисплей не поддерживает работу с окнами. Данное окно является «родительским» для остальных окон. Если очищается окно с номером «0» или выводится текст в это окно, очищаются все «дочерние» окна.

При создании окна задается его область на экране и один из режимов вывода текста. В дальнейшем, при выводе информации в созданное окно, передается только текст сообщения DisplayText. При необходимости, созданное окно можно очистить, командой ClearText.

После того, как окно стало ненужным, его можно уничтожить с помощью функции DestroyWindow.

При необходимости, можно вывести на экран Дисплея аппаратно-реализуемую заставку (рекламный текст, логотип), используя метод DisplayLogo.

## 4. Настройки

### 4.1. Общие настройки

В этом разделе описываются настройки общие для всех моделей данного типа оборудования. Данные настройки подлежат обязательной реализации в любом драйвере внешнего оборудования.

#### DefaultTimeOut

Описание:

Значение таймаута выполнения метода по умолчанию в секундах. Если таймаут, преданный в параметрах какого-либо метода, равен нулю, драйвер использует в качестве таймаута значение данной настройки.

Тип	Допустимые значения	Значение по умолчанию	Представление
Number	10..65535	120	Таймаут выполнения метода по умолчанию (в сек.)

#### EventLogEnabled

Описание:

Данная настройка предназначена для включения режима записи диагностической информации в журнал событий, либо лог-файл. Используется в целях отладки и диагностики проблем с драйвером устройства. Если установлено значение True, драйвер вызывает функцию обратного вызова [WriteLog](#) в те моменты, когда это необходимо (определяется разработчиком). Значение по умолчанию для данной настройки – False

Тип	Допустимые значения	Значение по умолчанию	Представление
Boolean	True/False	False	Запись диагностической информации

### 4.2. Специфические настройки

В этом разделе документа описываются настройки, специфичные для описываемого типа оборудования.

#### Rows

Количество строк дисплея. Только для чтения

Тип	Допустимые значения	Значение по умолчанию	Описание
Number	1...1000	-	Количество строк

#### Columns

Количество символов в строке дисплея. Только для чтения

Тип	Допустимые значения	Значение по умолчанию	Описание
Number	1...1000	-	Количество символов в строке

## 5. Методы

### 5.1. Служебные методы

В данном разделе описаны методы, обеспечивающие взаимодействие клиентского приложения и драйвера на начальных и конечных этапах цикла использования оборудования. Данные методы подлежат обязательной реализации в любом драйвере.

#### Init

**Синтаксис:**

*Init (ApplicationRef: COM-object, ErrorDescription: String): Boolean*

**Описание:**

Метод-конструктор объекта драйвера. При выполнении данного метода происходит начальная инициализация драйвера. Метод вызывается единожды, сразу после создания объекта. Любые другие методы объекта драйвера могут быть вызваны только после успешного выполнения метода Init. Если метод вернул значение False, объект драйвера должен быть уничтожен, оставив систему в исходном состоянии. Если метод выполнен успешно, то с объектом драйвера можно работать как с объектом-абстрактом. Например получить список настроек по умолчанию, вызвав метод [GetSettings](#).

**Параметры:**

*ApplicationRef: COM-object*

Ссылка на интерфейс IDispatch приложения-клиента. Используя данную ссылку, драйвер может вызывать у приложения функции обратного вызова (после выполнения метода Open).

*ErrorDescription: String*

Если метод возвращает False, в данном параметре содержится описание ошибки.

**Возвращаемое значение:**

True – метод выполнен успешно. False – в противном случае. В случае, если метод вернул результат False, дальнейшая работа с драйвером невозможна и данный объект драйвера должен быть уничтожен

#### Open

**Синтаксис:**

*Open (DeviceID: String, ErrorDescription: String): Boolean*

**Описание:**

Завершает инициализацию объекта драйвера и присваивает ему уникальный идентификатор экземпляра. В процессе выполнения данного метода, драйвер не должен захватывать какие-либо неразделяемые ресурсы (например, коммуникационные порты или файлы), и не должен выполнять никаких реальных взаимодействий с устройством. Драйвер проверяет наличие необходимых ему для работы ресурсов (файлов библиотек, их версий и т.п.) и возвращает Ложь, если проверки обнаружили невозможность полноценного функционирования драйвера. После успешного выполнения метода допускается вызов следующих методов драйвера: [ShowSettingsDlg](#), [CheckHealth](#), [Enable](#), [Disable](#). Все остальные методы будут доступны после включения устройства (метод Enable).

После выполнения метода драйвер может использовать [функции обратного вызова](#).

**Параметры:**

*DeviceID: String (GUID)*

Уникальный идентификатор экземпляра оборудования. Идентификатор необходим при использовании функций обратного вызова.

*ErrorDescription: String*

Если метод вернул False, данный параметр содержит описание ошибки.

**Возвращаемое значение:**

True – метод выполнен успешно. False – в противном случае. В случае, если метод вернул результат False, дальнейшая работа с драйвером невозможна. Необходимо вызвать метод [Close](#) и уничтожить объект драйвера.

#### Close

**Синтаксис:**

Close (): Boolean

**Описание:**

Метод-деструктор драйвера. Метод должен вызываться непосредственно перед уничтожением объекта драйвера. При выполнении данного метода, драйвер устройства должен освободить все захваченные ресурсы, и очистить ссылку на интерфейс IDispatch приложения, переданную в параметрах метода [Init](#). Никакие другие методы драйвера не могут быть вызваны после вызова Close

**Параметры:**

Нет

**Возвращаемое значение:**

True

## GetDeviceInfo

**Синтаксис:**

GetDeviceInfo (InputParameters: Undefined, ResultData: SafeArray, TimeOut: Number): Boolean

**Описание:**

Метод предназначен для получения статической информации о модели устройства. Допустимо вызывать данный метод в любой момент, даже у неинициализированного объекта (то есть до метода [Init](#))

**Параметры:**

*InputParameters: Undefined*

Не используется. Необходимо передавать пустое значение.

*ResultData: SafeArray*

При успешном исполнении метода данный параметр содержит массив SafeArray с информацией о модели устройства. В случае ошибки выполнения метода, параметр содержит массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут ожидания результата выполнения метода. (см. [TimeOut](#))

**Возвращаемое значение:**

True – метод выполнен успешно. False – в противном случае

**Массив ResultData:**

Индекс SafeArray	Имя	Тип	Описание
0	TypeName	String	Наименование типа оборудования
1	ModelID	String	GUID модели оборудования
2	ModelName	String	Наименование модели
3	MajorVersion	Number	Мажор-версия системы (контроль совместимости на строгое равенство)
4	MinorVersion	Number	Минор-версия системы (совместимость снизу-вверх)
5	BuildVersion	Number	Текущая версия драйвера (билд), никак не контролируется системой
6	ProcessorName	String	Программный идентификатор драйвера. Используется для создания объекта драйвера
7	ProcessorType	Number	Тип драйвера: 1 = COM объект Win32, 2 = COM объект Win64, 3 = Native API Win32, 4 = Native API Win64, 5 = Native API Linux32, 6 = Native API Linux64
8	IsSettingsDlgExist	Boolean	Наличие собственной формы настроек (см. описание метода <a href="#">ShowSettingsDlg</a> )
9	Description	String	Подробное описание модели
10	DeveloperName	String	Разработчик
11	CI_email	String	Контактная информация
12	CI_www	String	Контактная информация

13	CI_Phones	String	Контактная информация
14	CI_address	String	Контактная информация

## 5.2. Общие методы

Перечисленные в данном разделе методы являются обязательными для реализации в любом драйвере любого типа оборудования, соответствующего данному стандарту. Совокупность общих и специфических методов называется «публикуемыми» методами. Публикуемые методы доступны для вызова из любого места приложения, в то время, как служебные методы должен использовать менеджер оборудования. Вызов любого публикуемого метода возможен только у полностью инициализированного объекта драйвера (см. описание метода [Open](#)) за исключением методов [GetSettings](#), [GetDeviceInfo](#) и [ShowSettingsDlg](#), которые допустимо вызывать у объекта-абстракта. Входные и выходные параметры публикуемых методов всегда помещаются в одномерные массивы `SafeArray`. Все публикуемые методы всегда имеют три параметра:

*InputParameters: SafeArray*

Одномерный массив `SafeArray` с входными параметрами метода, либо Неопределено (`Undefined`) когда входные параметры отсутствуют.

*ResultData: SafeArray*

Одномерный массив `SafeArray` с результатами выполнения метода, либо Неопределенно (`Undefined`), если возвращаемые параметры отсутствуют. В случае, если метод возвращает `False`, в данном параметре содержится массив, содержащий код и описание ошибки (см. описание массива [ErrorInfo](#))

*TimeOut: Number*

Определяет максимальное время ожидания выполнения метода в секундах. Имеются два зарезервированных предопределенных значения:

0 – время исполнения метода не задано. Драйвер должен использовать значение настройки [DefaultTimeOut](#).

-1 – время исполнения метода не ограничено.

**Массив `ErrorInfo`:**

Индекс в <code>SafeArray</code>	Имя	Тип	Описание
0	<code>ErrorCode</code>	Number	Код ошибки, регламентированный стандартом (см. документ «Коды ошибок и их диапазоны.doc»)
1	<code>ErrorDescription</code>	String	Описание ошибки оборудования
2	<code>ExtData</code>	<code>SafeArray</code> / String / Number / DateTime / Boolean	Дополнительные данные или <code>Undefined</code> . Тип параметра определяется типом оборудования и вызываемым методом. Необязательный

### GetSettings

**Синтаксис:**

*GetSettings (InputParameters: Undefined, ResultData: SafeArray, TimeOut: Number): Boolean*

**Описание:**

Получение списка значений текущих настроек устройства. Если метод вызывается до первого вызова метода `SetSettings`, возвращаются значения настроек устройства по умолчанию, заданные разработчиком драйвера.

**Параметры:**

*InputParameters: Undefined*

Не используется. Необходимо передавать пустое значение.

*ResultData: SafeArray*

Массив значений настроек [Settings](#). В случае ошибки выполнения метода, параметр содержит массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут ожидания результата выполнения метода. (см. [TimeOut](#))

**Возвращаемое значение:**

True – метод выполнен успешно. False – в противном случае.

**Массив ResultData:**

Индекс SafeArray	Имя	Тип	Описание
0	<a href="#">Settings</a>	SafeArray	Массив значений настроек устройства

**Массив Settings**

Двумерный массив значений настроек устройства. Индекс старшего измерения (строки массива) изменяется в диапазоне от 0 до N (N = количество настроек - 1). Индекс младшего измерения (колонок массива) изменяется в диапазоне от 0 до 6

Индекс SafeArray	Имя	Тип	Описание
(0..N, 0)	ID	String	Идентификатор настройки. Идентификатор должен начинаться с буквы. Допустимо использование только латинских символов.
(0..N, 1)	Value	Number / String / DateTime / Boolean	Значение настройки
(0..N, 2)	Type	String	Строковое представление типа значения настройки. Допустимые варианты: «Number» «String» «DateTime» «Boolean»
(0..N, 3)	Default	Number / String / DateTime / Boolean	Значение настройки по умолчанию
(0..N, 4)	ReadOnly	Boolean	Если равно True, пользователь не может менять значение настройки
(0..N, 5)	Name	String	Пользовательское представление настройки
(0..N, 6)	Description	String	Описание настройки

**SetSettings****Синтаксис:**

*SetSettings (InputParameters: SafeArray, ResultData: SafeArray, TimeOut: Number): Boolean*

**Описание:**

Метод предназначен для установки новых значений настроек устройства. Допустимо вызывать данный метод до вызова метода [Open](#). Метод не должен вызываться приложением, если устройство находится в состоянии «Включено» (см. метод [Enable](#)).

**Параметры:**

*InputParameters: SafeArray*

Массив значений настроек [Settings](#).

*ResultData: SafeArray либо Undefined*

Возвращаемые параметры отсутствуют. В случае ошибки выполнения метода, параметр содержит массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут ожидания результата выполнения метода. (см. [TimeOut](#))

**Возвращаемое значение:**

True – метод выполнен успешно. False – в противном случае

**Массив InputParameters:**

Индекс SafeArray	Имя	Тип	Описание
0	<a href="#">Settings</a>	SafeArray	Массив значений настроек устройства

**ShowSettingsDlg****Синтаксис:**

*ShowSettingsDlg (InputParameters: Undefined, ResultData: SafeArray, TimeOut: Number): Boolean*

**Описание:**

При выполнении данного метода, драйвер отображает в модальном режиме форму для интерактивного изменения настроек устройства. Допускается отсутствие собственной формы настроек при реализации драйвера. В этом случае, метод возвращает False и ошибку с кодом 1003 (см. [ErrorInfo](#)), а клиентское приложение отображает собственную универсальную форму для редактирования настроек устройства.

**Параметры:**

*InputParameters: Undefined*

Не используется. Необходимо передавать пустое значение.

*ResultData: SafeArray*

При отсутствии у драйвера формы настроек, возвращается [ErrorInfo](#) с кодом ошибки 1003. Если пользователь отказался от внесения изменений (нажал кнопку «Отмена») возвращается [ErrorInfo](#) с кодом ошибки 201. В случае, если пользователь нажал кнопку «ОК» и настройки были успешно применены, содержит значение Undefined

*TimeOut: Number*

Таймаут ожидания результата выполнения метода. Значение данного параметра игнорируется

**Возвращаемое значение:**

True – метод выполнен успешно, настройки изменены и применены новые значения настроек.

False – изменения настроек не произошло.

**CheckHealth****Синтаксис:**

*CheckHealth (InputParameters: Undefined, ResultData: SafeArray, TimeOut: Number): Boolean*

**Описание:**

Метод используется для проверки работоспособности устройства. Вызов данного метода может приводить к исполнению длительных операций, поэтому в клиентском приложении не рекомендуется использовать данный метод в обычном цикле использования оборудования. Стандарт никак не регламентирует действия драйвера при выполнении данного метода. Состав производимых проверок определяется разработчиком драйвера.

**Параметры:**

*InputParameters: Undefined*

Не используется. Необходимо передавать пустое значение.

*ResultData: Undefined либо SafeArray*

Возвращаемые параметры отсутствуют. В случае ошибки выполнения метода, параметр содержит массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут ожидания результата выполнения метода. Значение данного параметра игнорируется

**Возвращаемое значение:**

True – устройство исправно и работоспособно. False – в противном случае

**Enable****Синтаксис:**

*Enable (InputParameters: Undefined, ResultData: SafeArray, TimeOut: Number): Boolean*

**Описание:**

Включение устройства. Метод должен быть вызван перед началом использования оборудования, до вызова любого специфического метода. При выполнении данного метода драйвер проводит все возможные проверки на готовность устройства к работе и только в случае их успешного завершения, возвращает результат True. Если драйверу требуется для работы монополюсный захват каких-либо ресурсов (например, коммуникационный порт), он должен выполнить его при выполнении данного метода.

**Параметры:**

*InputParameters: Undefined*

Не используется. Необходимо передавать пустое значение.

*ErrorInfo: SafeArray либо Undefined*

Возвращаемые параметры отсутствуют. В случае ошибки выполнения метода, параметр содержит массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут ожидания результата выполнения метода. (см. [TimeOut](#))

**Возвращаемое значение:**

True – устройство включено и готово к работе. False – в противном случае.

**Disable****Синтаксис:**

*Disable (InputParameters: Undefined, ResultData: SafeArray, TimeOut: Number): Boolean*

**Описание:**

Выключение устройства. Если драйвером были монополюно захвачены какие-либо ресурсы (например, коммуникационный порт), он должен освободить их при выполнении данного метода.

**Параметры:**

*InputParameters: Undefined*

Не используется. Необходимо передавать пустое значение.

*ErrorInfo: SafeArray*

Возвращаемые параметры отсутствуют. В случае ошибки выполнения метода, параметр содержит массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут ожидания результата выполнения метода. (см. [TimeOut](#))

**Возвращаемое значение:**

True – устройство успешно выключено. False – в противном случае

### 5.3. Специфические методы

Перечисленные в данном разделе методы являются специфическими методами типа оборудования. Любой из этих методов может быть вызван только у включенного устройства (см. Enable).

**CreateWindow****Синтаксис:**

*CreateWindow (InputParameters: SafeArray, ResultData: Undefined, TimeOut: Number): Boolean*

**Описание:**

Создает окно (область) для вывода текста. Окна номеруются, начиная с 1, и уничтожаются после выключения дисплея или командой [DestroyWindow](#).

**Параметры:**

*InputParameters: SafeArray*

Содержит параметры создаваемого окна.

*ResultData: Undefined*

Не используется. В случае ошибки - стандартный массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут исполнения команды (в секундах). См. [TimeOut](#)

**Возвращаемое значение:**

True – метод выполнен успешно. False – в противном случае

**Массив InputParameters**

Индекс SafeArray	Имя	Тип	Описание
0	Row	Number	Номер начальной строки. Верхняя строка имеет номер 0
1	Column	Number	Номер начальной колонки (символа). Крайний левый символ имеет номер 0
2	Width	Number	Ширина окна в символах
3	Height	Number	Высота окна в строках
4	WindowNum	Number	Номер окна. Допустимые значения 1..100
5	ScrollingMode	Boolean	Если равно True текст выводится в режиме прокрутки
6	ScrollingType	Number	Направление прокрутки: 0 – влево (по умолчанию), 1 - вправо 2 – вверх, 3 – вниз
7	BlinkMode	Boolean	Если равно True текст выводится в мигающем режиме

## DestroyWindow

### Синтаксис:

*CreateWindow (InputParameters: SafeArray, ResultData: Undefined, TimeOut: Number): Boolean*

### Описание:

Уничтожает ранее созданное окно. Если окно с указанным номером не существует или WindowNum равен 0 (окно, созданное по умолчанию невозможно уничтожить), метод возвращает ошибку

### Параметры:

*InputParameters: SafeArray*

Содержит номер удаляемого окна.

*ResultData: Undefined*

Не используется. В случае ошибки - стандартный массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут исполнения команды (в секундах). См. [TimeOut](#)

### Возвращаемое значение:

True – метод выполнен успешно. False – в противном случае

### Массив InputParameters

Индекс SafeArray	Имя	Тип	Описание
0	WindowNum	Number	Номер окна. Допустимые значения 1..100

## DisplayLogo

### Синтаксис:

*DisplayLogo (InputParameters: Undefined, ResultData: Undefined, TimeOut: Number): Boolean*

### Описание:

Выводит на экран аппаратную заставку дисплея. Не обязательный. Параметры аппаратной заставки настраиваются с помощью специализированного ПО или из формы настроек драйвера и не регламентируются в стандарте.

### Параметры:

*InputParameters: Undefined*

Не используется. Необходимо передавать пустое значение.

*ResultData: Undefined*

Не используется. В случае ошибки - стандартный массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут исполнения команды (в секундах). См. [TimeOut](#)

### Возвращаемое значение:

True – метод выполнен успешно. False – в противном случае

## DisplayText

### Синтаксис:

*DisplayText (InputParameters: SafeArray, ResultData: Undefined, TimeOut: Number): Boolean*

### Описание:

Выводит текст в указанное окно. При этом затираются все данные, содержащиеся в окне до вызова метода. Если окна накладываются друг на друга, то в пересекаемой области выведется текст окна, для которого данный метод был вызван последним. Если текст выводится в окно с номером «0», очищаются все остальные окна.

### Параметры:

*InputParameters: SafeArray*

Содержит текст, для вывода на экран дисплея и номер окна.

*ResultData: Undefined*

Не используется. В случае ошибки - стандартный массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут исполнения команды (в секундах). См. [TimeOut](#)

### Возвращаемое значение:

True – метод выполнен успешно. False – в противном случае

### Массив InputParameters

Индекс SafeArray	Имя	Тип	Описание
0	Text	String	Текст, выводимый на экран дисплея. Может содержать символы перехода строки.
1	WindowNum	Number	Номер окна, в которое будет выведен текст

### ClearText

#### Синтаксис:

*ClearText (InputParameters: SafeArray, ResultData: Undefined, TimeOut: Number): Boolean*

#### Описание:

Очищает текст, который был в окне до вызова метода. Если номер окна равен «0», то очищаются все окна.

#### Параметры:

*InputParameters: SafeArray*

Содержит номер окна для очистки.

*ResultData: Undefined*

Не используется. В случае ошибки - стандартный массив [ErrorInfo](#)

*TimeOut: Number*

Таймаут исполнения команды (в секундах). См. [TimeOut](#)

#### Возвращаемое значение:

True – метод выполнен успешно. False – в противном случае

### Массив InputParameters

Индекс SafeArray	Имя	Тип	Описание
0	WindowNum	Number	Номер очищаемого окна

## 6. Функции обратного вызова

В этом разделе описаны функции, подлежащие обязательной реализации в менеджере управления оборудованием любого клиентского приложения. Все функции предназначены для вызова их драйверами устройств. При вызове метода Init, в его параметрах передается ссылка на интерфейс IDispatch приложения-клиента. Используя эту ссылку, драйвер может вызывать функции обратного вызова, реализованные в приложении. Для типа оборудования "Дисплей покупателя" не предполагается использование таких функций обратного вызова как DataEvent, TaskState, MessageDlg, InputDataDlg.

### GetAppProperty

#### Синтаксис:

*GetAppProperty (SourceID: String, Name: String, Value: String): Boolean*

#### Описание:

Функция предназначена для получения свойств системы и приложения-клиента.

#### Параметры:

*SourceID: String (GUID)*

Идентификатор устройства, сгенерировавшего событие

*Name: String*

Наименование требуемого свойства. Допустимые значения:

- «ModelsFolder» - Путь к каталогу моделей оборудования
- «DevicesFolder» - Путь к каталогу экземпляров оборудования
- «Language» - Язык локализации приложения

*Value: String*

В этом параметре возвращается значение свойства. Для свойства «Language» возвращается двухсимвольный идентификатор языка («ru», «en», «ua» и т. д.)

#### Возвращаемое значение:

True – функция выполнена успешно, указанное свойство найдено. False – в противном случае

## ErrorEvent

### Синтаксис:

*ErrorEvent (SourceID: String, ErrorName: String, Description: String, ExtData: SafeArray): Boolean*

### Описание:

Функция предназначена для передачи приложению информации о возникновении ошибочного состояния или аварийной ситуации в устройстве

### Параметры:

*SourceID: String (GUID)*

Идентификатор устройства сгенерировавшего событие

*ErrorName: String*

Строка с наименованием типа ошибки. Драйвер оборудования может генерировать ошибки разных типов.

*Description: String*

Строка с кратким описанием возникшей ошибки или аварийной ситуации в устройстве

*ExtData: SafeArray*

Массив, содержащий дополнительные данные об ошибке. Необязательный.

### Возвращаемое значение:

True – функция выполнена успешно, ошибка обработана. False – в противном случае

## WriteLog

### Синтаксис:

*WriteLog (SourceID: String, Event: String, Text: String, Type: Number): Boolean*

### Описание:

Функция предназначена для передачи диагностической и отладочной информации приложению-клиенту. Приложение должно сохранять данную информацию в журнале (лог-файле). Данную функцию можно вызывать в любой момент – на усмотрение разработчика драйвера.

Рекомендуется использовать её для диагностики проблем и ошибок при работе с устройством.

Если настройка драйвера [EventLogEnabled](#) имеет значение False, функция вызываться не должна.

### Параметры:

*SourceID: String (GUID)*

Идентификатор устройства, сгенерировавшего событие

*Event: String*

Наименование события. Произвольная строка. По наименованию событий можно фильтровать/группировать записи в журнале.

*Text: String*

Описание события. Произвольная строка

*Type: Number*

Тип события. Допустимые значения: 0 – Информация; 1 – Ошибка; 2 - Отладка

### Возвращаемое значение:

True – функция выполнена успешно, информация записана приложением в журнал. False – в противном случае