

**ADK**  
**Application Developer Kit**  
**Printable Online Help**





**ADK**  
**Application Developer Kit**  
**Printable Online Help**

Copyright © Tektronix. All rights reserved. Licensed software products are owned by Tektronix or its subsidiaries or suppliers, and are protected by national copyright laws and international treaty provisions.

Tektronix products are covered by U.S. and foreign patents, issued and pending. Information in this publication supersedes that in all previously published material. Specifications and price change privileges reserved.

TEKTRONIX and TEK are registered trademarks of Tektronix, Inc.

MSDN is registered trade mark of Microsoft.

Online help version: 00

(4/17/2012)

077-0693-00 is the printable PDF file of online help 076-0277-00.

### **Contacting Tektronix**

Tektronix, Inc.  
14150 SW Karl Braun Drive  
P.O. Box 500  
Beaverton, OR 97077  
USA

For product information, sales, service, and technical support:

- In North America, call 1-800-833-9200.
- Worldwide, visit [www.tektronix.com](http://www.tektronix.com) to find contacts in your area.

# Table of Contents

## Introduction

Documentation .....	1
Using online help .....	1
General help functions .....	1
Conventions .....	2
Technical support .....	3

## Getting started

ADK overview .....	5
What is new in this release .....	6
System requirement .....	6
Application directories and usage .....	7

## Operating basics

Basic interfaces .....	9
INormalizedVector interface .....	9
IFastFrame interface .....	9
ISettings interface .....	10
IResult interface .....	11
IResultCollection interface .....	11
DPOJET Measurement Plug-in .....	13
Math Plugins	
Using math plugins .....	14
Writing math plugins .....	14
MATLAB custom functions .....	19
Using the basic Function interface to create MATLAB functions .....	19
Using a Class to create MATLAB functions .....	20
DataAccess public interface .....	21
TekScope Class .....	22
ITekScopeDataSource interface .....	22
ITekScopeDataAccess interface .....	23
ITekScopeData interface .....	23
License interface .....	23
ITektronixLicenseFactory Class .....	24
ITektronixLicense interface .....	25
ITektronixLicenseDetails interface .....	25
ITekScopeLicense interface .....	25
ITekScopeLicenseDetails interface .....	26

## Table of Contents



---

LicenseStatusChangeDelegate delegate .....	26
LicenseType enumeration .....	26
OptionStatus enumeration .....	27
LicenseAcquireStatus enumeration .....	27
LicenseReleaseStatus enumeration .....	27
GUI Toolkit .....	28
Project templates.....	30

## Index

## Documentation

The following table lists the documentation that is available for the product and shows where to find it.

Product documentation item	Purpose	Location
Online Help	Provides developers guide related to ADK toolsets.	 www.Tektronix.com
User Manual (adapted from the online help)	Provides developers guide related to ADK toolsets.	 www.Tektronix.com

## Using online help

Select Help from the menu to open the help file. You can also find an electronic copy of the help file in the Documents directory.

**Tables of contents (TOC) tab.** Organizes the Help into book-like sections. Select a book icon to open a section; select any of the topics listed under the book.

**Index tab.** Enables you to scroll a list of alphabetical keywords. Select the topic of interest to show the appropriate help page.

**Search tab.** Allows a text-based search.

Follow these steps:

1. Type the word or phrase you want to find in the search box. If the word or phrase is not found, try the Index tab.
2. Choose a topic in the lower box, and then select the Display button.

## General help functions

- Select the Print button from the Help topics menu bar to print a topic.
- To return to the previous window, select the Back button.
- Use hyperlinks to jump from one topic to another.
- If the back button is grayed out or a jump is not available, choose the Help Topics button to return to the originating help folder.

## Conventions

This documents use conventions to help readers distinguish source code from language elements, from keyboard sequences, and so on. Document conventions are similar to MSDN and the following table clarifies the conventions used.

**Table 1: Typographic conventions**

Convention	Description	Example
Monospace	Indicates source code, code examples, input to the command line, application output, code lines embedded in text, and variables and code elements.	<code>Class</code>
Bold	Indicates most predefined programming elements, including namespaces, classes, delegates, objects, interfaces, methods, functions, macros, structures, constructors, properties, events, enumerations, fields, operators, statements, directives, data types, keywords, exceptions, non-HTML attributes, configuration tags, registry keys, subkeys, and values.	<b>Path</b> class <b>Resolve</b> method
Italic	Indicates placeholders, most often method or function parameters; these placeholders represent information that must be supplied by the implementation or the user.	<i>context</i> parameter
Capital letters	Indicates the names of keys and key sequences.	ENTER CTRL+R
Plus sign	Indicates a combination of keys. For example, ALT+F1 means to hold down the ALT key while pressing the F1 key.	ALT+F1



## Technical support

Tektronix values your feedback on our products. To help us serve you better, please send us your suggestions, ideas, or comments on your application or oscilloscope.

When you contact Tektronix Technical Support, please include the following information (be as specific as possible):

### General information

- All instrument model numbers.
- Hardware options, if any.
- Probes and other accessories used.
- Your name, company, mailing address, phone number, and FAX number.
- Please indicate if you would like to be contacted by Tektronix about your suggestion or comments.

### Application specific information

- Software version number of firmware and all other software installed on the instrument.
- Description of the problem such that technical support can duplicate the problem.
- If possible, source code of the program you are developing.
- If possible, save the setup files for all the instruments used and the application.
- If possible, save the waveform on which you are performing the measurement as a .wfm file.

Forward the information to technical support using one of these methods:

- E-mail: [techsupport@tektronix.com](mailto:techsupport@tektronix.com)
- FAX: (503) 627-5695



## ADK overview

The Application Developer Kit (ADK) provides tools for oscilloscope users and third parties to create custom applications for Tektronix oscilloscopes. ADK provides developers fast access to waveform data, the ability to add measurements and to develop well integrated applications using Tektronix user interface controls. ADK also provides developers access to use the oscilloscope license mechanism to optionally license the application.

ADK toolsets are developed on the .Net 4.0 platform and provide developers a wide range of choices of development environment including Visual Studio and MATLAB. Users can develop applications using a variety of software development tools which can interface with .Net 4.0 APIs. ADK toolsets provide the following functionality:

1. **DataAccess Interface:** The DataAccess interface provides the ability to directly access the oscilloscope waveform data in the fastest way possible. This interface provides read access for all analog channels and math channels in sample, average and fast-frame modes along with waveform metadata; the DataAccess interface also provides read access to all digital channels.

With this unique DataAccess interface, the user developed program executes automatically within the oscilloscope acquisition sequence cycle; meaning when new acquisition data is available the oscilloscope automatically executes the users program. The oscilloscope acquisition cycle would allow the user program to complete the operation on the waveform data before starting with the next acquisition.

This is a relatively different programming paradigm where it enables users better separation of program functionality related to waveform data operation and other program functionality.

2. **DPOJET Measurement Plug-in:** DPOJET measurement plug-in for *DPOJET Jitter and Eye Diagram Analysis* enables developers to add user defined standards under the DPOJET standard group and provides the ability to add new measurements under the user defined standard tab.

The user added measurements to DPOJET are treated the same way as inbuilt DPOJET measurement, all other functionality such as report generation and results statistics are automatically available for the user added measurements. (Similar to the DataAccess interface, the user measurement executes automatically within the oscilloscope acquisition sequence cycle.)

3. **MATH Plug-in:** MATH plug-in enables developers to add user defined MATH functions. The user defined MATH functions are treated the same way as inbuilt MATH functions. (Similar to the DataAccess interface, the user defined MATH functions execute automatically within oscilloscope acquisition sequence cycle.)
4. **GUI toolkit:** ADK provides access to selective oscilloscope user-interface controls directly in the Visual Studio development environment. Using these user-interface controls, developers can develop applications with the identical look and feel as the oscilloscope user-interface. ADK also provides seamless integration of user developed applications with the oscilloscope user-interface within the "Analyze" drop-down menu of the TekScope user-interface.

5. **License Interface:** Using the license interface developers can use the oscilloscope license mechanism to optionally license the application. This provides the same functionality (such as free trials), that is available for Tektronix-licensed applications.
6. **Project Templates:** Various Visual Studio project templates are available as part of ADK. These project templates show usage of ADK toolsets and are integrated with the Visual Studio development environment. These project templates are expected to serve as a starting point for developers to build applications. Visual Studio project templates are available for the DataAccess Interface, DPOJET Measurement plug-in, MATH plug-in, GUI toolkit and License Interface and supported in the Visual Basic and C# languages.

## What is new in this release

The current release replaces the earlier released *Beta* version of the ADK with the following feature improvements.

1. Simplified DataAccess interface with support for On-Demand access.
2. Simplified DPOJET Measurement plug-in interface.
3. Added MATH Plug-in support.
4. GUI application with improved integration with the TekScope user-interface.
5. Added new VisualStudio templates.

## System requirement

ADK is supported on the Tektronix DPO/DSA/MSO 5K/7K/70K series of Real-time oscilloscopes with the Windows 7 (64-bit) operating system.

Before installing ADK on your oscilloscope, please verify the following software are installed on the oscilloscope:

- Oscilloscope Firmware
- DPOJET Jitter And Eye Diagram Analysis
- Visual Studio 2010 (Professional / Premium / Ultimate edition)

## Application directories and usage

Following table lists the default directory names and their usage.

Directory path	Usage
C:\Program Files\Tektronix\ADK	Application installation path and contains the application files.
<vs path>\Microsoft Visual Studio 10.0\Common7\IDE\Extensions\Tektronix	Includes ADK shipped Project Templates
C:\Users\Public\Tektronix\ADKApps\<AppName>	Installation path for TekScope GUI integrated applications <sup>1</sup>
C:\Users\Public\Tektronix\Plugins\DPOJET	Installation path for DPOJET measurement plug-in <sup>1</sup>
C:\Users\Public\Tektronix\Plugins\Math	Installation path for MATH plug-in <sup>1</sup>

<sup>1</sup> The installation in the Public folder will make the application available for all users, the application can also be installed in "C:\Users\<current user>\Tektronix\" which will make the application available only for the current user.



## Basic interfaces






This section provides an overview of base element interfaces, classes and enumeration which are frequently used while developing applications using the Tektronix ADK tool.

### INormalizedVector interface



Namespace: `Tek.Scope.Support`

Assembly: `ScopeSupportBase` (in `ScopeSupportBase.dll`)

#### Properties:

	Name	Description
	Count	The number of elements in the array.
	Data[Int64]	The normalized data element at the specified location.
	Horizontal	The horizontal section.
	SourceName	Returns the name of the source for this vector.
	Vertical	The vertical section.

#### Methods:



	Name	Description
	Commit	Tells the underlying class(es) that this set of changes is complete. This allows any housekeeping associated with a consistent state to be done.
	ToArray	Returns the data as an array of type double.

### IFastFrame interface






Namespace: `Tek.Scope.Support`

Assembly: `ScopeSupportBase` (in `ScopeSupportBase.dll`)


#### Properties:

	Name	Description
	Count	The number of elements in the array.
	CurrentFrame	Current Frame

**Properties:**

	<b>Name</b>	<b>Description</b>
	Data[Int64]	The normalized data element at the specified location.
	FrameCount	Total number of frames in FastFrame data
	Horizontal	The horizontal section.
	SourceName	Returns the name of the source for this vector.
	Vertical	The vertical section.

**Methods:**





	<b>Name</b>	<b>Description</b>
	Commit	Tells the underlying class(es) that this set of changes is complete. This allows any housekeeping associated with a consistent state to be done.

**ISettings interface**



Namespace: Tek.Scope.Support

Assembly: ScopeSupportBase (in ScopeSupportBase.dll)

**Properties:**






	<b>Name</b>	<b>Description</b>
	Names	Iterate through symbol names
	Item[String]	Sets/Gets symbols in the table.
	IsAborting	Used to indicate that the current operation should be aborted.
	IsEmpty	Returns true if the contents are empty.

**Methods:**

	<b>Name</b>	<b>Description</b>
	Clear	Resets Symbol Table to empty.
	Contains(String)	Returns true if the passed argument is a keyword.



**Methods:**






	<b>Name</b>	<b>Description</b>
	Contains(String, String)	Returns true if the passed arguments are a keyword and a corresponding value.
	GetNumber(String, String, Double)	This method returns the number value associated with the specified symbol name. If the name does not exist, then the passed default value is used.
	GetString(String, String, String)	This method returns the string value associated with the specified symbol name. If the name does not exist, then the passed default value is used.
	ReadCSV(String)	Reads the named CSV file into the current Attribute state.
	writeCSV(String)	Writes the current state to the named CSV file.

## IResult interface



Namespace: Tek.Scope.Support

Assembly: ScopeSupportBase (in ScopeSupportBase.dll)

**Properties:**

	<b>Name</b>	<b>Description</b>
	Begin	Returns the begin location.
	Duration	Returns the width of this item.
	End	Returns the end location.
	Focus	Returns the focus of this item. This value must be between Begin and End.
	Value	Value. Typically the results of some calculation. But if there is not a traditional result, then the Duration is used.

**Methods:**

	<b>Name</b>	<b>Description</b>
	CompareTo(Object)	Compares the current instance with another object of the same type.
	CompareValue(Object)	Compare value against the passed object. A return value of null means the objects were not comparable.

## IResultCollection interface




Namespace: Tek.Scope.Support

Assembly: ScopeSupportBase (in ScopeSupportBase.dll)





**Properties:**

	<b>Name</b>	<b>Description</b>
	Begin	Returns the begin location.
	Count	Returns the count of items in the collection.
	Duration	Returns the width of this item.
	End	Returns the end location.
	Focus	Returns the focus of this item. This value must be between Begin and End.
	HUnits	The units for the horizontal section.
	Item[Int32]	Returns the item at the specified index.
	Maximum	Returns the maximum value.
	Mean	Returns the average of the Values.
	Minimum	Return the minimum value.
	Name	Accesses the name of this item.
	PeakToPeak	Peak2Peak measurement.
	SourceName	Accesses the source name of this item.
	StandardDeviation	Returns the Standard Deviation.
	VUnits	The units for the vertical section.

**Methods:**

	<b>Name</b>	<b>Description</b>
	Add(Double, Double, Double, Double)	Add a value to the collection.
	Add(IResult)	Add a value of type IResult to the collection.
	clear()	Clear the collection.

**Methods:**

	Name	Description
	Commit()	Tells the underlying class(es) that this set of changes is complete. This allows any housekeeping associated with a consistent state to be done.
	Contains(IResult)	Check whether the specified IResult item exists in the collection.
	Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.
	GetEnumerator	Allows foreach and linq to work with this interface.

## DPOJET Measurement Plug-in

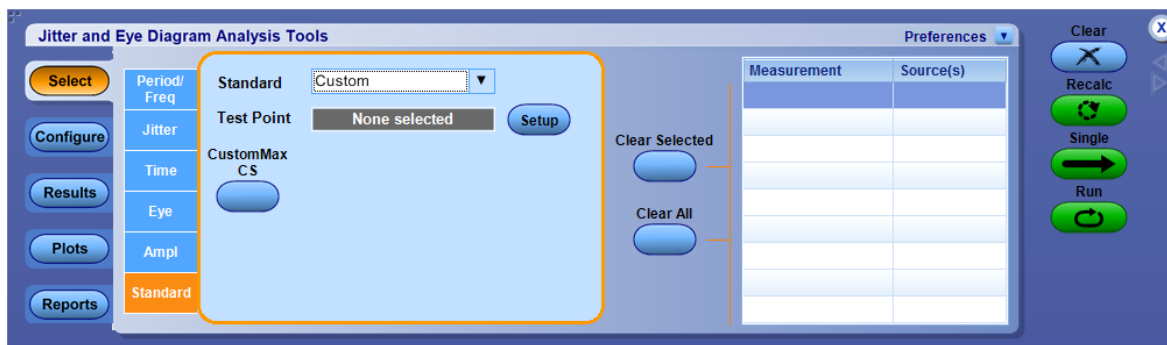
DPOJET Measurement Plug-in interface under `Tektronix.Scope.Applications.DpoJet` namespace provides base element function attribute which are consumed by client applications.

To define any measurement as a DPOJET plug-in, the following two steps need to be implemented:

- 1. Define the method attribute as DPOJETMeasurement as shown below, with the optional StandardName
- 2. The method signature should be as shown below

```
[DPOJETMeasurement(StandardName = "Custom")]
public IResultCollection CustomMaxCS(IList<INormalizedVector> inputWFMs, ISettings measSettings, out ResultStatus status)
```

The measurement added using this plug-in appears in DPOJET under standard tab as shown below. For more details refer the ADK templates available in VisualStudio under Tektronix.



## Using math plugins

Math plugins extend the current built-in math system on your instrument. When the TekScope application starts, it scans `C:\Users\Public\Tektronix\Plugins\Math` and `C:\Users\<current user ID>\Tektronix\Plugins\Math` for .NET libraries and loads any tagged functions into the math system. For a library to load, it must have the word `math`, `meas` or `plugin` in its name. If a plugin library is placed in one of the folders after the TekScope application has started, the plugin will not be available until the application is restarted. Plugins are only loaded when TekScope starts up to avoid performance impact.

Once the instrument has started, the plugin is used in the math system like any built in function. For example, if one of the libraries provides two functions, `MyAdd` and `MyMultiply`, each of which takes two vector inputs, the following math definitions are valid:

- `Math1=MyAdd(Ch1, Ch2)`
- `Math1=MyMultiply(Ch1+Ch2, Ch3)`
- `Math2=MyAdd(Ref1, Math1)`
- `Math1=MyMultiply(Ch1, Ch2)+Inv(Ch3)`
- `Math1=MyMultiply(Avg(Ch1), Ch2)`

The math system generates standard syntax errors if there is an issue with a math equation using a plugin, such as a missing parenthesis. If there is an error with the plugin itself, math reports a plugin specific error. This may be that the plugin is not valid because it has mismatched input and output types or that an argument specified in the equation does not match the type expected by the plugin (for example, a string was expected but a `FastFrame` waveform was supplied).

## Writing math plugins

You can write plugins in any .NET language. The following examples use C#. When writing a plugin, reference the `ScopeSupportBase.dll` and `TekScriptingEngine.dll` system assemblies. Additionally, tag all plugins with the C# attribute `[Math]` if they are to be loaded into the system.

Six different classes are used inside plugins: `INormalizedVector`, `IFastFrame`, `IWaveformDB`, `IString`, `ISettings` and `IRange`. The first two, `INormalizedVector` and `IFastFrame`, are waveform vector types used by the instrument. The third, `IWaveformDB` is a pixmap waveform. `IStrings`, `ISettings` and `IRange` are used to pass additional information into a plugin.

### Waveform types

All waveform classes contain a member called `SourceName`. This is a string that contains the symbol name of the waveform source. If the source is a channel, math or reference waveform, the `SourceName` will be `Ch<x>`, `Math<n>` or `Ref<n>`, respectively. If the waveform is an intermediate, the `SourceName`

will be `Intermediate0`, if it is the output waveform or `Intermediate<n>` if it's an input vector, where `<n>` corresponds to which argument it is. Examples of intermediates are:

- `Math1=MyAdd(Ch1, Ch2)*Ch3`: The output of `MyAdd` is an intermediate because it still needs to be multiplied by `Ch3` before being put into `Math1`. The `SourceName` for the output waveform will be `Intermediate0`.
- `Math1=MyAdd(Ch1*Ch2, Ch3)`: The first input is an intermediate because two channels are being multiplied together. Its `SourceName` will be `Intermediate1`.
- `Math1=MyAdd(Ch1, Ch2/Ch3)`: The second input is an intermediate because two channels are being divided. Its `SourceName` will be `Intermediate2`.

`INormalizedVector` is the basic waveform type used by TekScope. The length of the vector is found in the `Count` member of the class. Array indices are used to access values inside the vector:

```
INormalizedVector output;
INormalizedVector input1;
for (long i = 0; i < input1.Count; i++)
    output[i] = input1[i];
```

`IFastFrame` is built on top of `INormalizedVector`. An `IFastFrame` is a grouping of `INormalizedVectors`. `IFastFrames` are generated when `FastFrame` is enabled on the instrument. The number of frames is stored in the member called `FrameCount`. You can iterate through the frames by setting the `CurrentFrame` (note: frames are 1 counted so you should iterate from 1 to `FrameCount`). Once you set the `CurrentFrame`, you use the `IFastFrame` the same as an `INormalizedVector`.

```
IFastFrame output;
IFastFrame input1;
IFastFrame input2;
for (long f = 1; f <= output.FrameCount; f++)
{
    input1.CurrentFrame = f;
    input2.CurrentFrame = f;
    output.CurrentFrame = f;

    for (long i = 0; i < output.Count; i++)
        output[i] = input1[i] * input2[i];
}
```

The final waveform type is `IWaveformDB` (waveform database or DPO data). This type essentially tracks hits in a visual manner. The higher the value that is stored into a point relative to the other values stored, the brighter the color will be. Unlike an `INormalizedVector` or an `IFastFrame`, the `IWaveformDB` goes both horizontally and vertically across the screen. The horizontal length is in `Horizontal.Count` and the vertical height is in `Vertical.Count`. To access a point inside the `IWaveformDB`, use double array indices where the vertical position comes before the horizontal position:

```
IWaveformDB output;
long hCount = output.Horizontal.Count;
long vCount = output.Vertical.Count;
for (long hh = 0; hh < hCount - 1; hh++)
{
    Parallel.For(0, vCount, vv =>
```

```

    {
        output[vv, hh] = output[vv, hh + 1];
    });
}

```

If the output type of a plugin is `IWaveformDB`, when the plugin is called the output waveform is populated with the values the plugin returned the last time it was called. If this is the first time the plugin was called, all of the values inside the `IWaveformDB` will be zero (displays as clear).

The advanced user can change the vertical scale and position, as well as the horizontal scale and spacing by setting `Vertical.Scale`, `Vertical.Position`, `Horizontal.Scale`, and `Horizontal.Spacing` in the `IWaveformDB`.

## IString

A plugin may use an unlimited number of `IStrings` as input. An `IString` is simply a string and is used to pass meta data or additional information into a plugin.

## ISettings: scope settings

You can access information about instrument settings through `ISettings`. Settings for the math target, as well as the input and output waveforms, are put into the dictionary. However, if the waveform is an intermediate, no setting information is available. Examples of intermediates are:

- `MATH1=Add(Ch1*Ch2, Ch3)`: `Ch1*Ch2` is an intermediate
- `MATH1=Add(Ch1, Ch2)*Ch3`: The output of `Add` is an intermediate

To look up information in `ISettings`, you need to know the waveform name. For input or output waveforms, this is put into the `SourceName` field. This is either `Ch<n>`, `Math<n>`, `Ref<n>` or `Intermediate<n>` if it is a channel, math, reference, or intermediate waveform respectively. To find the name of the target math, look up `MathTarget` in `ISettings` (a string of the form `Math<n>` is returned).

Once you have the waveform name, you can look up the following information:

- Vertical scale (“VScale”)
- Vertical offset (“VOffset”)
- Vertical position (“VPosition”)
- Vertical units (“VUnits”)
- Horizontal scale (“HScale”)
- Horizontal offset (“HOffset”)
- Horizontal position (“HPosition”)
- Horizontal units (“HUnits”)

Additionally, the following information about the target math is available:

- LPCT
- MPCT

- HPCT
- HYSTPCT

The recommended method for looking up information is to use the `ISettings::GetNumber(string sourceName, string name, double defaultValue)` and `ISettings::GetString(string sourceName, string name, double defaultValue)` functions. For example, to get the name of the target math:

```
string mathTarget = settings.GetString("", "mathTarget", "");
```

Or to get the vertical scale for an input:

```
double scale = settings.GetNumber(!string.IsNullOrEmpty(input1.SourceName) ?  
input1.SourceName : "", "vscale", double.NaN);
```

Note: Strings are not case sensitive, so `VSCALE` and `vscale` return the same information.

You can also use the array operators to look up information, but if the information is not in the dictionary you may end up with null objects:

```
string mathTarget = settings["mathTarget"];  
double scale = settings["ch1.vscale"];
```

This may be useful for debug purposes however.

### **IRange: gating information**

Currently, the `IRange` parameter will always be null. It is in place to support future features.

### **Plugin rules**

Plugins require at least one and at most two vector inputs (either `INormalizedVector` or `IFastFrame`). If two vector inputs are used by the plugin, they both must be the same type. Additionally, plugins may take an unlimited number of `IStrings` as input. When using `IString` inputs, it is important to remember that the entire math equation, when typed into the editor, is limited to 128 characters.

Plugins can generate an `INormalizedVector`, `IFastFrame` or `IWaveformDB` waveform as output. If the output type is `INormalizedVector`, the vector input(s) must also be `INormalizedVector`. For `IFastFrame` and `IWaveformDB` outputs, the inputs can be `INormalizedVector` or `IFastFrame`.

If `FastFrame` is turned off and the plugin takes `IFastFrames` as input, the plugin is given `IFastFrames` that consist of only one frame. How the plugin behaves when `FastFrame` is turned on depends on the output waveform type. If the output is an `INormalizedVector`, which takes only `INormalizedVector` as input, the plugin is called once per frame and the math system handles iterating through all of the frames. If the plugin produces `IFastFrame` or `IWaveformDB` it is called once per `FastFrame` acquisition. When the plugin takes as input an `IFastFrame`, it is given all of the frames at once. If the plugin takes an `INormalizedVector` as input, the plugin will only see the first frame which is put into the `INormalizedVector`. The rest of the frames in the `FastFrame` are not seen by the plugin.

### Summary of valid plugin signatures and behavior

Output Type	Vector Input Type	FastFrame Off	FastFrame On
INormalizedVector	INormalizedVector	Called once per acq	Called once per frame
IFastFrame	INormalizedVector	Called once per acq	First frame put into INormalizedVector
IFastFrame	IFastFrame	Called once per acq; Only contains one frame	Called once per FastFrame
IWaveformDB	INormalizedVector	Called once per acq	First frame put into INormalizedVector
IWaveformDB	IFastFrame	Called once per acq; Only contains one frame	Called once per FastFrame

### Example plugins

If the Application Developer Kit has been installed on the instrument, example plugins can be found in Microsoft Visual Studio. When you create a new project, choose Visual C#->Tektronix->Math to access the examples.

### Create a plugin

To create a plugin, you need to use either one of our Visual Studio templates or create a new project using the .NET language of your choice. To create a C# plugin from scratch, you first need to create a new, empty C# project. Once you create your project, add references to ScopeSupportBase, TekScriptingEngine, System.Data, System.Data.DataSequence, System.XML and System.Xml.Linq. In your code, use System, System.Collections.Generic, System.Linq, System.Text, and Tek.Scope.Support.

There are no requirements for the name of the class or namespace the plugins use. In this example the namespace is MyMathPlugins and the class name is MyMath. The plugin function is a public, static function inside the class with the math attribute. We call our plugin Add, and it takes two INormalizedVectors as input and produces an INormalizedVector.

```
namespace MyMathPlugins
```

```
{
```

```
    class MyMath
```

```
    {
```

```
        // Add(<wfm>, <wfm>):
```

```
        // This plugin adds two INormalizedVector inputs
```

```
        [Math]
```

```
        public static void Add(ISettings settings, IRange gate, INormalizedVector output, INormalizedVector input1, INormalizedVector input2)
```

```
        {
```

```
            // We only want to add up to the shorter input length
```



```
if (input1.Count < input2.Count)
{
    output.Count = input1.Count;
    output.Horizontal.Spacing = input1.Horizontal.Spacing;
    output.Horizontal.ZeroIndex = input1.Horizontal.ZeroIndex;
}
else
{
    output.Count = input2.Count;
    output.Horizontal.Spacing = input2.Horizontal.Spacing;
    output.Horizontal.ZeroIndex = input2.Horizontal.ZeroIndex;
}
// Add two inputs together
for (long i = 0; i < output.Count; i++)
    output[i] = input1[i] + input2[i];
}
}
```

When compiling the project, make sure the target platform is Any CPU. Once the DLL is compiled (release or debug, both work), it should be placed in the appropriate directory on the instrument. Restart the instrument application, and the plugin is ready to use like any existing math operator.

## MATLAB custom functions

The Custom Analysis Interface for use with MATLAB provides two options for writing MATLAB custom analysis functions: a basic function interface and a more advanced class-based interface. Both types are available in demo form on the instrument in C:\Users\Public\Tektronix\Plugins\Math\MATLAB.

### Using the basic Function interface to create MATLAB functions

The function interface uses a simple signature: `function [ output ] = exampleProcessingFunction( firstTime, varargin )`. Your function should take two inputs: a Boolean that indicates whether or not this is the first time the function has been called and a variable length array. Put the results that you want the instrument to display into the output variable of the same length as the input array.

If this is the first time the function has been called as part of the math expression, `firstTime` is true. In this case, the `varargin` array consists of the record length and the sample rate:

```
recordLength = varargin{1};
sampleRate = varargin{2};
```

and the output is expected to be true:

```
output = true;
```

If you have any one time processing, such as filter creation, do this when `firstTime` is true. If you want to have variables from a previous execution of the function available, you should mark the variables persistent. See the examples for more information. Note that persistent variables are automatically cleared before the first execution of the function.

If this is not the first time the function has been called, then `firstTime` is false and `varargin` will consist of the waveform inputs. The math expression may have one or two waveform inputs to the MATLAB function:

```
input1 = varargin{1};
if numel(varargin) == 2
    input2 = varargin{2};
end
```

From here you can do any computations you want and put the results into the output. Your MATLAB function can use any features available in MATLAB or installed toolboxes to perform its calculations. Note that the output must have the same length as the inputs. If your output vector is shorter, zero out the remaining points.

## Using a Class to create MATLAB functions

An advanced user can create a MATLAB custom analysis function by subclassing `instrument.integration.AlgorithmDefinition`. The `waterfall.m` class shows an example of this type of custom analysis function. This is more complex than using a function but allows more control of behavior than the basic function capability. By using a subclass, you can implement custom tear-down behaviors, such as closing plots automatically when an analysis function is no longer being called.

Your class should be a subclass of `instrument.integration.AlgorithmDefinition`:

```
classdef myClass < instrument.integration.AlgorithmDefinition
```

There are four methods your class may need to implement: a constructor, a destructor, a process, and a `stopProcessingHook`. The default destructor and `stopProcessingHook` from the superclass may be sufficient for your class. However, at minimum you need to implement a constructor and the process function:

```
classdef myClass < instrument.integration.AlgorithmDefinition
    methods
        function obj = myClass(sampleRate,pointsPerRecord)
obj@instrument.integration.AlgorithmDefinition(sampleRate,pointsPerRecord);
        end
        function [result] = process(obj,varargin)
```

```

        result = varargin{1};
    end
end
end
end

```

The constructor is called the first time the analysis function is used in a specific math expression on the instrument. Here anything that requires setup before processing data, such as plots, should be configured. The parent constructor can be called from your custom construct if necessary (the constructor for `AlgorithmDefinition` sets up some basic settings information about the instrument).

The process function is the main workhorse of the class. This function computes results and returns data to the instrument. Like the custom analysis functions created using the basic function interface, the length of the results returned by your process function should match the length of the input.

The `stopProcessingHook` function is called when the analysis function is no longer in use by the instrument. This may happen when you edit or clear a math function, or in other situations when the instrument math system expects that the state should be reset. For example, if your analysis function is displaying a plot, this would be the time to close the plot.


Finally, the destructor is called when the analysis function is no longer in use. Typically you call the `stopProcessingHook` function from here to aid in cleanup.

## DataAccess public interface



The `Tektronix.Scope.Support` namespace contains classes and interfaces which allow an application to read and write the data to the base oscilloscope under Win 7 64 bit operating system.

The following table shows the classes and interfaces available in the `Tektronix.Scope.Support` namespace.




### Classes:

	Name	Description
	TekScope	Provides the instance of the Data Source for clients to connect.


### Interfaces:

	Name	Description
	ITekScopeDataSource	Provides functionality to allow clients to connect to the oscilloscope to access the data.
	ITekScopeDataAccess	This interface allows the client to access the data available with the oscilloscope. This interface is available to the clients in the callback function to access the data.

**Interfaces:**

	Name	Description
	ITekScopeData	Provides access to meta-data of the waveform data produced by the oscilloscope.
	INormalizedVector	Provides access to waveform data.
	IFastFrame	Provides access to Fast Frame data.

**Delegates:**



	Name	Description
	DataAvailableForAccess	Represents the signature of the method which is invoked for the clients to access the data from the Data Source.

## TekScope Class

Namespace: Tektronix.Scope.Support

Assembly: TekScopeDataNetInterfaces (in TekScopeDataNetInterfaces.dll)

**Methods:**




	Name	Description
	GetDataSource	Returns ITekScopeDataSource instance for connecting with Data Source.
	GetDataSourceImmediate	Returns ITekScopeDataSource instance for connecting with Immediate Data Source.

## ITekScopeDataSource interface

Namespace: Tektronix.Scope.Support

Assembly: TekScopeDataNetInterfaces (in TekScopeDataNetInterfaces.dll)

**Methods:**


	Name	Description
	Connect	Method to connect to the Data store. Return true when connection successful.
	Disconnect	Method to Disconnect from Data Source.
	ReinitiateDataAccess	Method to request a callback to access the data from the data source.

## ITekScopeDataAccess interface

Namespace: Tektronix.Scope.Support

Assembly: TekScopeDataNetInterfaces (in TekScopeDataNetInterfaces.dll)

### Methods:




	Name	Description
	GetData	Method to access the data available in Data Source. <b>Return Value</b> <i>ITekScopeData</i> Returns non null value if data is available.

## ITekScopeData interface

Namespace: Tektronix.Scope.Support

Assembly: TekScopeDataNetInterfaces (in TekScopeDataNetInterfaces.dll)

### Properties:


	Name	Description
	CanWrite	Returns true if the data can be modified
	DataCounter	Returns long value. If this value is greater than the last time the code queried, it denotes new data.
	IsDataNew	Returns true if the data is new. Always check this flag before accessing the data.

ITekScopeData interface can be type cast to INormalizedVector, IFastFrame or IDigitalEvents to access the waveform data.





## License interface

License interface under Tektronix.Scope.License namespace provides the following base element interfaces, classes and enumeration which are consumed by client applications.


### Classes:

	Name	Description
	ITektronixLicenseFactory	Provides static factory method for ITektronixLicense Interface.





**Interfaces:**

	Name	Description
	<code>ITektronixLicense</code>	Provides basic interface for Tektronix license.
	<code>ITektronixLicenseDetails</code>	Provides basic interface for specifying details of Tektronix license.
	<code>ITekScopeLicense</code>	Provides detailed interface for oscilloscope license. Inherits from <code>ITektronixLicense</code> interface.
	<code>ITekScopeLicenseDetails</code>	Provides detailed interface for specifying details of oscilloscope license. Inherits from <code>ITektronixLicenseDetails</code> interface.

**Delegates:**

	Name	Description
	<code>LicenseStatusChangeDelegate</code>	Represents the method that will handle <code>LicenseStatusChangeNotifier</code> event raised by <code>ITekScopeLicense</code> .

**Enumerations:**


	Name	Description
	<code>LicenseType</code>	Specifies license type.
	<code>OptionStatus</code>	Specifies license option status.
	<code>LicenseAcquireStatus</code>	Specifies license acquire status.
	<code>LicenseReleaseStatus</code>	Specifies license release status.

**ITektronixLicenseFactory Class**

Namespace: `Tektronix.Scope.License`

Assembly: `TekScopeLicenseNetInterface` (in `TekScopeLicenseNetInterface.dll`)

**Methods:**


	Name	Description
	<code>getITektronixLicenseInterface</code> ( <code>string OptionName</code> )	Returns <code>ITektronixLicense</code> instance for specified <code>OptionName</code> .

## ITektronixLicense interface

Namespace: Tektronix.Scope.License

Assembly: TekScopeLicenseNetInterface (in TekScopeLicenseNetInterface.dll)

### Properties



	Name	Description
	LicenseDetails	Gets ITektronixLicenseDetails interface.

## ITektronixLicenseDetails interface

Namespace: Tektronix.Scope.License

Assembly: TekScopeLicenseNetInterface (in TekScopeLicenseNetInterface.dll)

### Properties:

	Name	Description
	OptionName	Gets Option Name.
	Type	Gets License Type.





## ITekScopeLicense interface

Namespace: Tektronix.Scope.License



Inherits from: ITektronixLicense

Assembly: TekScopeLicenseNetInterface (in TekScopeLicenseNetInterface.dll)



### Properties:

	Name	Description
	AcquireLicenseStatus	Gets License Acquire Status.
	AcquireLicenseStatusVerbose	Gets License Acquire Status Verbose details.
	OptionStatus	Gets Option Status.
	OptionStatusVerbose	Gets Option Status Verbose details.

**Properties:**

	Name	Description
	ReleaseLicenseStatus	Gets License Release Status.
	ReleaseLicenseStatusVerbose	Gets License Release Status Verbose details.

**Methods:**

	Name	Description
	AcquireLicense	Method to acquire the license.
	ReleaseLicense	Method to release the license.

**Events:**

	Name	Description
	LicenseStatusChangeNotifier	Occurs when Option Status changes.


**ITekScopeLicenseDetails interface**

Namespace: `Tektronix.Scope.License`

Inherits from: `ITektronixLicenseDetails`

Assembly: `TekScopeLicenseNetInterface` (in `TekScopeLicenseNetInterface.dll`)

**Properties:**

	Name	Description
	OptionNumber	Gets Option Number.

**LicenseStatusChangeDelegate delegate**

Namespace: `Tektronix.Scope.License`

Assembly: `TekScopeLicenseNetInterface` (in `TekScopeLicenseNetInterface.dll`)

**LicenseType enumeration**

Namespace: `Tektronix.Scope.License`

Assembly: `TekScopeLicenseNetInterface` (in `TekScopeLicenseNetInterface.dll`)



**Members:**

Name	Description
Custom	Reserved for future, not supported.
ScopeFixedOption	Fixed oscilloscope option.
ScopeFloatingOption	Floating oscilloscope option.
FlexLM	Reserved for future, not supported.

**OptionStatus enumeration**

Namespace: Tektronix.Scope.License

Assembly: TekScopeLicenseNetInterface (in TekScopeLicenseNetInterface.dll)

**Members:**

Name	Description
Available	Available for permanent, unconditional.
AvailableLimited	Available but limited, such as evaluation period, free trials, time limited and others.
Unavailable	Option either not enabled or evaluation expired.

**LicenseAcquireStatus enumeration**

Namespace: Tektronix.Scope.License

Assembly: TekScopeLicenseNetInterface (in TekScopeLicenseNetInterface.dll)

**Members:**

Name	Description
Success	License acquire operation successful.
Limited	License acquire operation successful, typically under OptionStatus AvailableLimited.
Failed	License acquire operation failed, typically under OptionStatus Unavailable.
uninitialized_OR_NotAcquired	Initial state or before license is acquired.

**LicenseReleaseStatus enumeration**

Namespace: Tektronix.Scope.License

Assembly: TekScopeLicenseNetInterface (in TekScopeLicenseNetInterface.dll)

**Members:**

Name	Description
Success	License release operation successful.
Failed	License release operation failed, typically under OptionStatus Unavailable.
Uninitialized_OR_NotReleased	Initial state or before license is released.

## GUI Toolkit

Following TekScope user controls are published as part of ADK and are integrated with Visual Studio development environment.

**Table 2: GUI Toolkit**







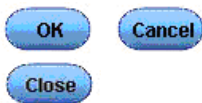






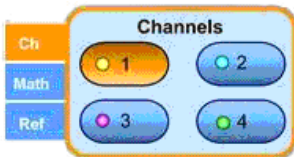



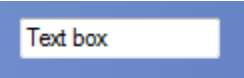
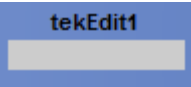
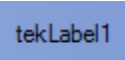
Control name	Image of control	Use of control
AppSequencerControl		The green button is an action button used for specific instances: <ul style="list-style-type: none"> <li>■ Recalc</li> <li>■ Single</li> <li>■ Run</li> </ul>
ControlWindowPanel		Tektronix blue colored blank panel
TekCheckButton		Shows an On state of a setting.
		Supports multiple selections.
TekRadioButton		Supports an exclusive selection. Only one of a set of radio buttons may be the selected button. Best used when there are only a limited number of selections possible- otherwise a dropdown makes better use of the UI real estate space. There is always one button of a set that is selected. By default, the first shown of a collection of radio buttons is selected.

Table 2: GUI Toolkit (cont.)

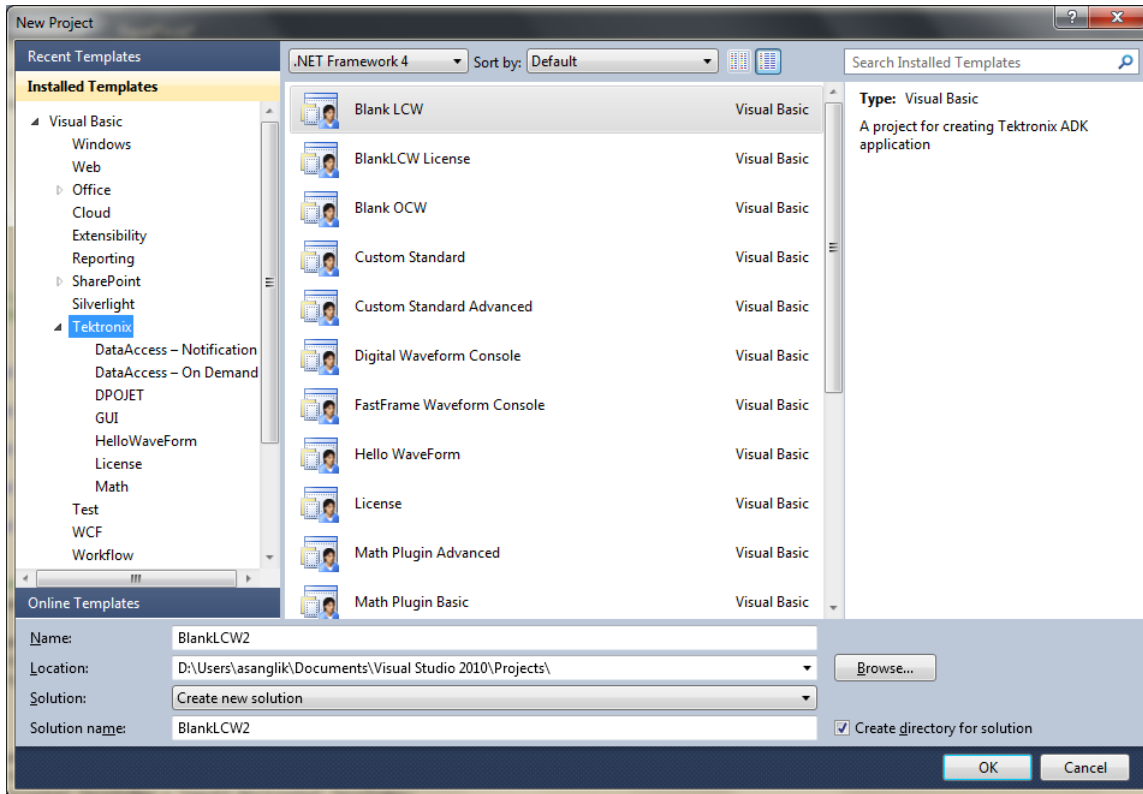
Control name	Image of control	Use of control
		Shows an Off state of a setting.
		Used to execute an action to take, such as "Save", or to exit out of a UI (OK/Cancel are used if a change can be made while Close is used if no change can be made/lost).
		Special instance of the Blue Button which opens up a Windows File Browser.
		Special instance of the Blue Button which navigates either to a previous or to the next UI in a flow-like set of UIs.
TekPushButton		Special instance of the Blue Button which signifies a Clear action.
		Is used to navigate to an OCW.
		Special instance of an aqua button which navigates to a plot UI.
TekDropDown		Supports an exclusive selection. Only one of the options in the dropdown options list may be selected. Makes the best use of UI real estate space, but the other options are not visible to the user until they click the dropdown. There is always one option of a set that is selected, and by default, the first shown. To force a user to specify an option, the default selection could be "Please Select...".
TekInputSelector		Specific instance of an internal CW tab that supports selection of sources.
TekPanel		Tektronix light blue colored blank panel with orange colored border.

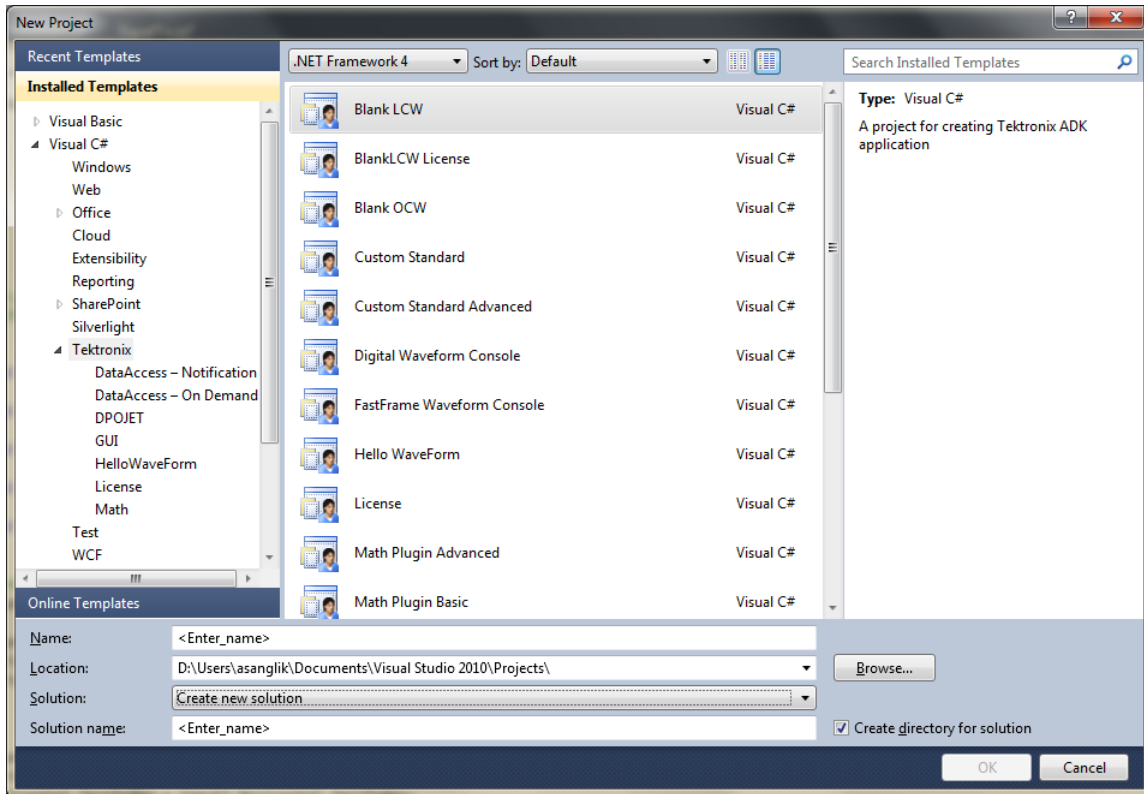
**Table 2: GUI Toolkit (cont.)**

Control name	Image of control	Use of control
TekTab		Tab driven UI, typically used with "ControlWindowPanel" or "TekPanel"
TekDisplay		Used for numeric or string input / output (as read only)
TekTextBox		Text box, is used for string input / output (as read only)
TekEdit		Similar to TekTextBox
TekLabel		Label

## Project templates

Various Visual Studio project templates are available as part of ADK. These project templates show usage of ADK toolsets and are integrated with the Visual Studio development environment. These project templates are expected to serve as a starting point for developers to build applications. Visual Studio project templates are available for the DataAccess Interface, DPOJET Measurement plug-in, MATH plug-in, GUI toolkit and License Interface and are supported in Visual Basic and C# languages.





# Index

## A

Application directories and usage, 7

## B

Basic interfaces, 9

## C

Class

ITektronixLicenseFactory, 24  
TekScope, 22

Conventions, 2

## D

DataContext public interface, 21

Delegate

LicenseStatusChangeDelegate, 26

Documentaiton, 1

DPOJET measurement plug-in, 13

## E

Enumeration

LicenseAcquireStatus, 27  
LicenseReleaseStatus, 27  
LicenseType, 26  
OptionStatus, 27

## G

Getting started, 5

GUI Toolkit, 28

## I

IFastFrame interface, 9

INormalizedVector interface, 9

Interface

DataContext public, 21

IFastFrame, 9

INormalizedVector, 9

IResult, 11

IResultCollection, 11

ISettings, 10

ITekScopeData, 23

ITekScopeDataAccess, 23

ITekScopeDataSource, 22

ITekScopeLicense, 25

ITekScopeLicenseDetails, 26

ITektronixLicense, 25

ITektronixLicenseDetails, 25

License, 23

IResult interface, 11

IResultCollection interface, 11

ISettings interface, 10

ITekScopeData interface, 23

ITekScopeDataAccess

interface, 23

ITekScopeDataSource

interface, 22

ITekScopeLicense interface, 25

ITekScopeLicenseDetails

interface, 26

ITektronixLicense interface, 25

ITektronixLicenseDetails

interface, 25

ITektronixLicenseFactory

class, 24

ITektronixLicenseFactory

Class, 24

## L

License interface, 23

LicenseAcquireStatus

enumeration, 27

LicenseReleaseStatus

enumeration, 27

LicenseStatusChangeDelegate

delegate, 26

LicenseType enumeration, 26

## M

Math plugins

using, 14

writing, 14

## N

New in this release, 6

## O

Online help, 1

using, 1

OptionStatus enumeration, 27

Overview, 5

## P

Plug-in

DPOJET measurement, 13

Project templates, 30

## S

System requirements, 6

## T

technical support, 3

Technical support, 3

TekScope class, 22

Templates, 30

## U

User manual, 1