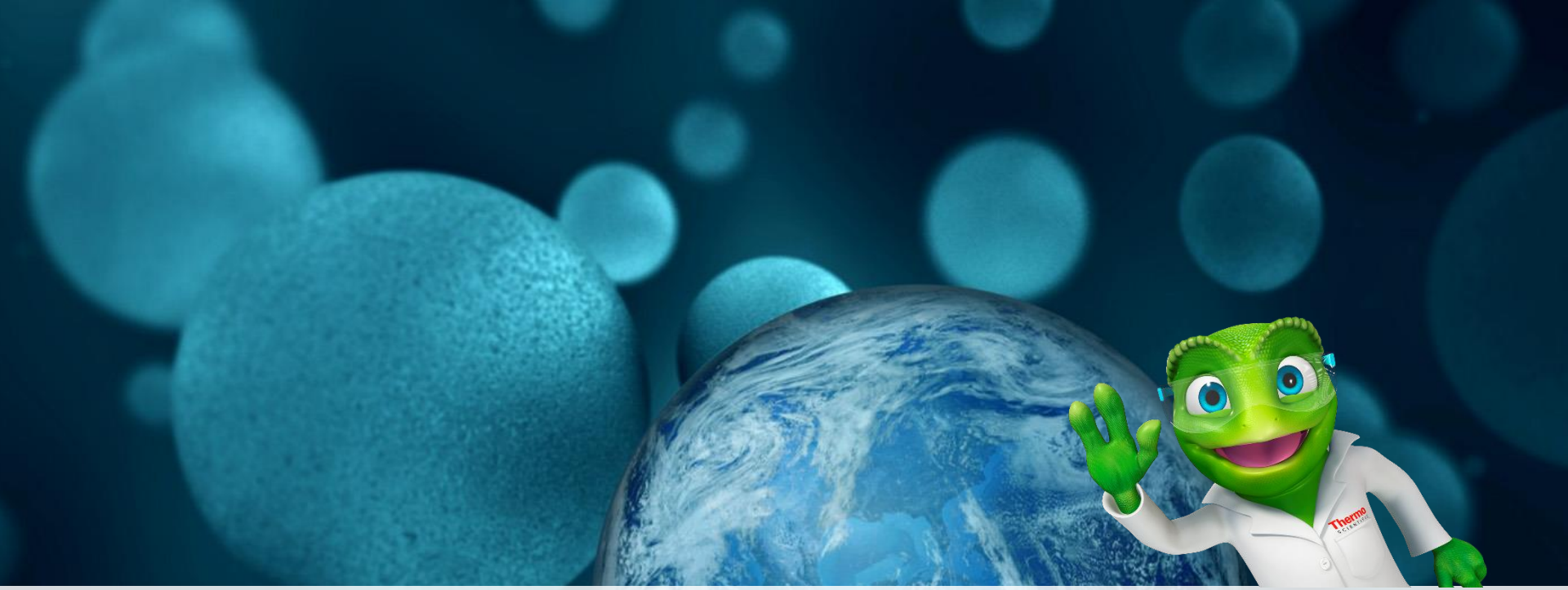


ThermoFisher SCIENTIFIC

Chromeleon DDK Development Training – Day3

Anton Kyosev, CMD Senior Staff Software Engineer
Yubo Dong, CMD Software Engineering Manager

Day 3 Morning Agenda

- Driver & Devices
  - Pump
  - AutoSampler
  - Detector
  - Channel

Device Driver Interfaces

IDevice

- Functions to create properties, commands creation
- Functions for special interfaces:

Pump

```
IFlowHandler CreateFlowHandler(ITypeDouble tFlow,
// Number of eluent components, that the pump can handle
                               int numberOfComponents,
// Number of digits for the eluent component percentage
                               int percentDigits)
```

AutoSampler

```
IInjectHandler CreateInjectHandler(ITypeDouble tInjectionVolume,
                                   ITypeInt    tInjectionPosition)
```

Pump Driver

Use `IDevice` to create flow handler

```
IFlowHandler m_FlowHandler = Device.CreateFlowHandler
```

Implement event handlers

```
m_FlowHandler.FlowNominalProperty.OnSetProperty
m_FlowHandler.FlowNominalProperty.OnSetProperty
m_FlowHandler.ComponentProperties[i].OnSetProperty
```

For gradient pumps

```
m_FlowHandler.FlowNominalProperty.OnSetRamp
m_FlowHandler.ComponentProperties[i].OnSetRamp
```

Add pressure property

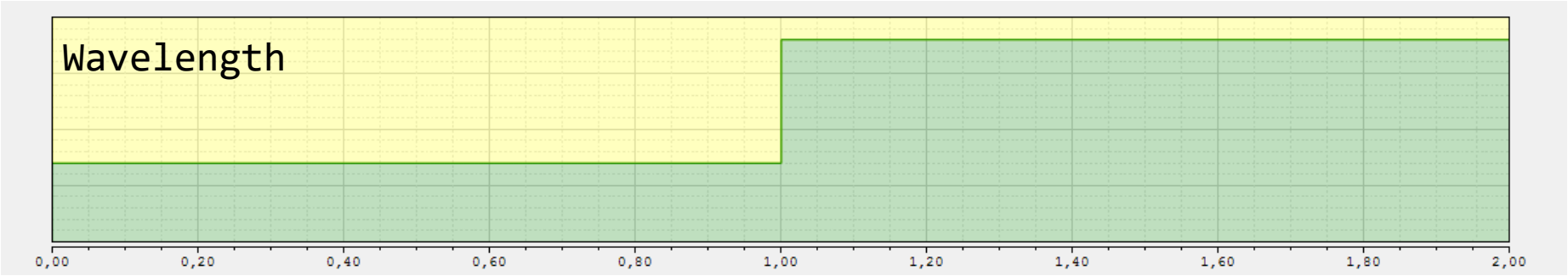
Example – CmDDKExamples**ExampleLCSystem**\Pump.cs

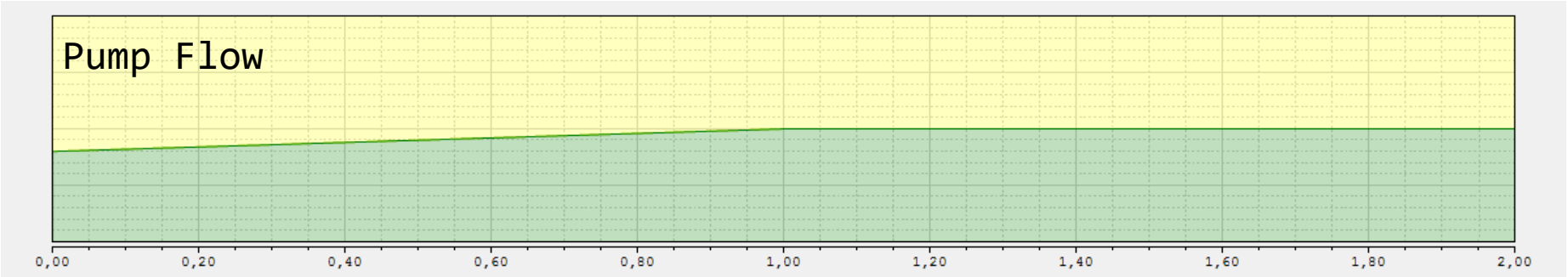
Pump Driver – Ramp Syntax

Consider the following 2 examples:

```
Time
0.000 Wavelength = 200        Flow = 0.4
1.000 Wavelength = 220        Flow = 0.5
2.000 End
```

The user expects the following behavior:





Pump Driver – Ramp Syntax

In order to specify which behavior is intended, you need to set `RampSyntax = true` for the `IProperty` that you have created for the Flow property.

```
m_FlowHandler.FlowNominalProperty.RampSyntax = true;
m_FlowHandler.FlowValueProperty  .RampSyntax = true;
```

The getter is called `IsRampSyntax.`

This must be set during the `IDriver.Init` function – when the symbols are defined. It is not possible to set this later on or dynamically.

Example – CmDDKExamples**TimeTableDriver**\TimeTableDevice.cs

Pump Driver – Ramp Syntax

```
Time
0.000 Wavelength = 200        Flow = 0.4
1.000 Wavelength = 220        Flow = 0.5
2.000 End
```

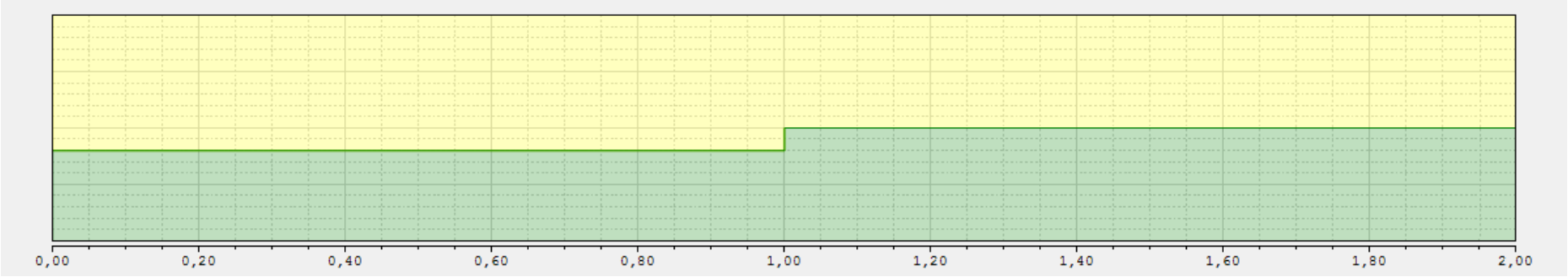
Based on whether `IsRampSyntax` is True or False, the method execution engine generates different events during method execution:

Execution Time	**IsRampSyntax False** (Wavelength)	**IsRampSyntax True** (Flow)
0.000	OnSetProperty() with args.NewValue = 200	OnSetRamp() with args.Start = 0.4 args.End = 0.5 args.Duration = 1.000
1.000	OnSetProperty() with args.NewValue = 220	OnSetProperty() with args.NewValue = 0.5
2.000		

Pump Driver – Ramp Syntax

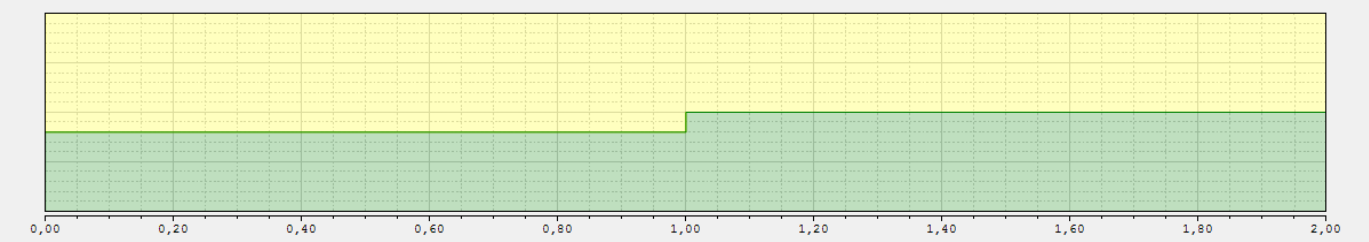
In the (rare) case that a step-like change for a property with ramping behavior is intended, the method can express this by

```
0.000 Flow = 0.4
1.000 Flow = 0.4
      Flow = 0.5
2.000 End
```



Pump Driver – Ramp Syntax

```
0.000 Flow = 0.4
1.000 Flow = 0.4
      Flow = 0.5
2.000 End
```



Execution Time	**IsRampSyntax True** (Flow)
0.000	OnSetRamp() with args.Start = 0.4 args.End = 0.4 args.Duration = 1.000
1.000	OnSetRamp() with args.Start = 0.4 args.End = 0.5 args.Duration = 1.000
1.001	OnSetProperty() with args.NewValue = 0.5
2.000	

AutoSampler Driver

```
IInjectHandler m_InjectHandler = CreateInjectHandler(
                                    ITypeDouble tInjectionVolume,
                                    ITypeInt    tInjectionPosition)
```

Have 2 properties and 1 command:

```
IIntProperty     m_InjectHandler.PositionProperty
IDoubleProperty  m_InjectHandler.VolumeProperty [µl]
ICommand         m_InjectHandler.InjectCommand
```

- The **Position** and **Volume** properties are initialized from the values entered in the Injection List at the beginning of an injection; i.e. before the corresponding instrument method is started .
- The **Inject** command will temporarily suspend method execution until the sampler driver has signalized that injection has taken place and analysis can be started.
- The **Position** property type must be defined appropriately so that it reflects the number and layout of available vials and trays in the sampler.
- `m_InjectHandler.NotifyInjectResponse()`

  `Have to` be called by the driver after successful injection. **Note**: The driver needs to call this as soon as possible after the injection valve has switched from Load to Inject. Don't wait until any post-switch operation (such as a wash or homing) is complete.

To abort an injection:

- `m_InjectHandler.NotifyMissingVial(string errorMessage)`

  To be called by the driver when the module has determined that there is no vial at the specified position. This will abort the current Injection (state changes to Interrupted), but sequence processing continues with the next Injection.

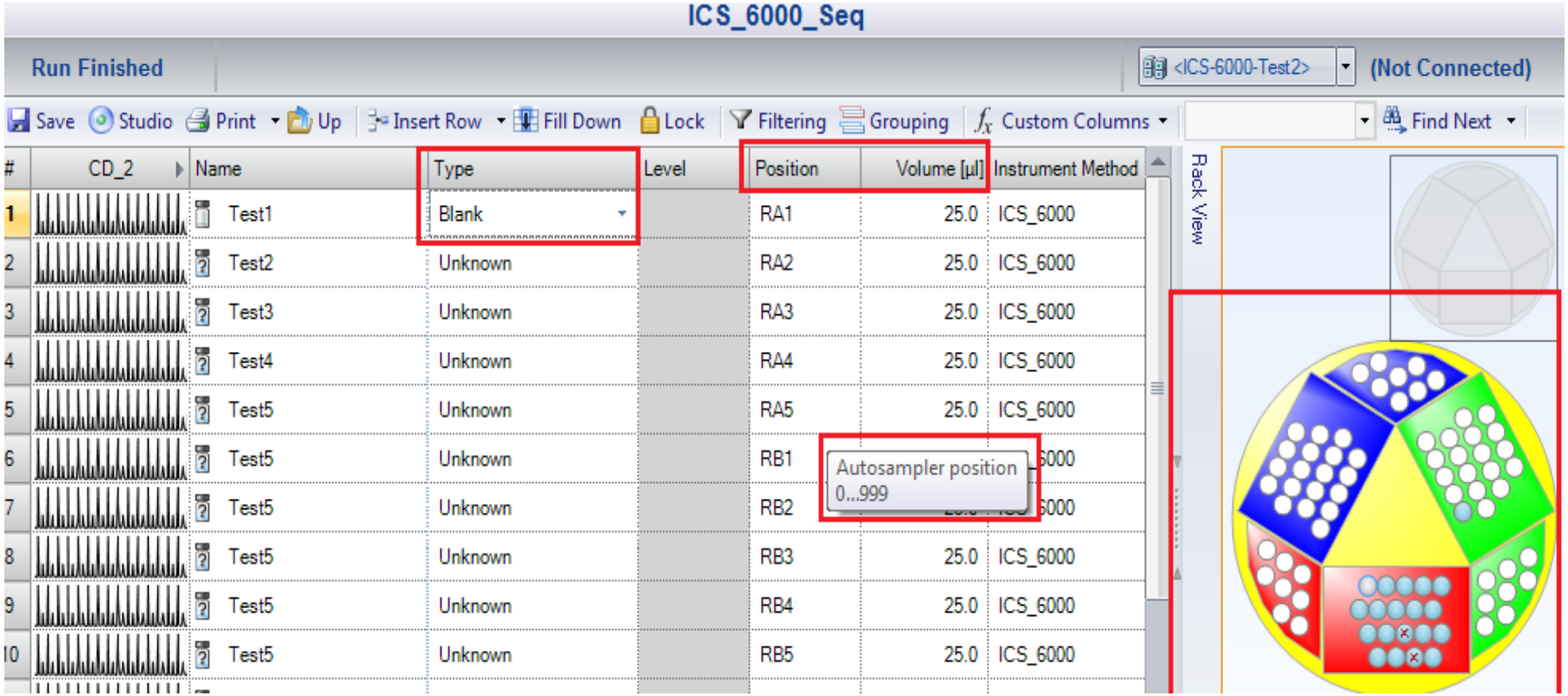
- `m_Device.AbortSample(string messageText)`

  Abort the current Injection. Use this for other fatal situations during injection.

Example
\CmDDKExamples**AutoSampler**\AutoSamplerDevice.cs

Autosampler Driver Interface

The Rack View is not available for all autosampler models. A rack view is only displayed if the selected instrument supports the rack visualization function and the selected instrument is connected to the Instrument Controller Service

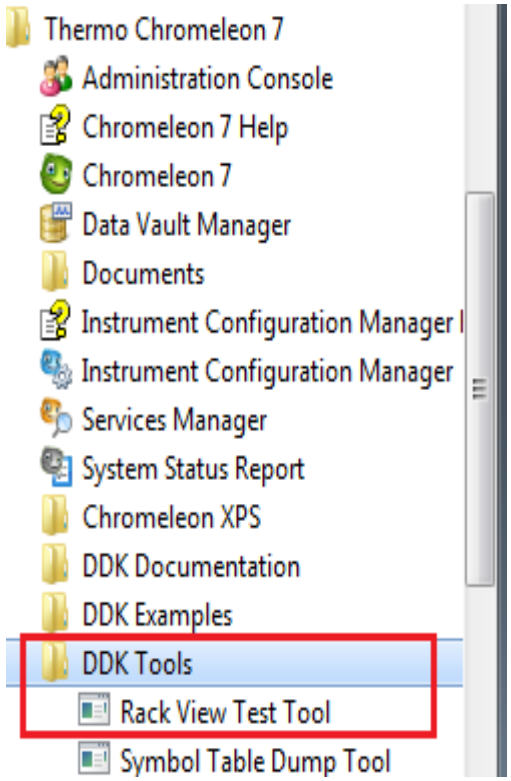


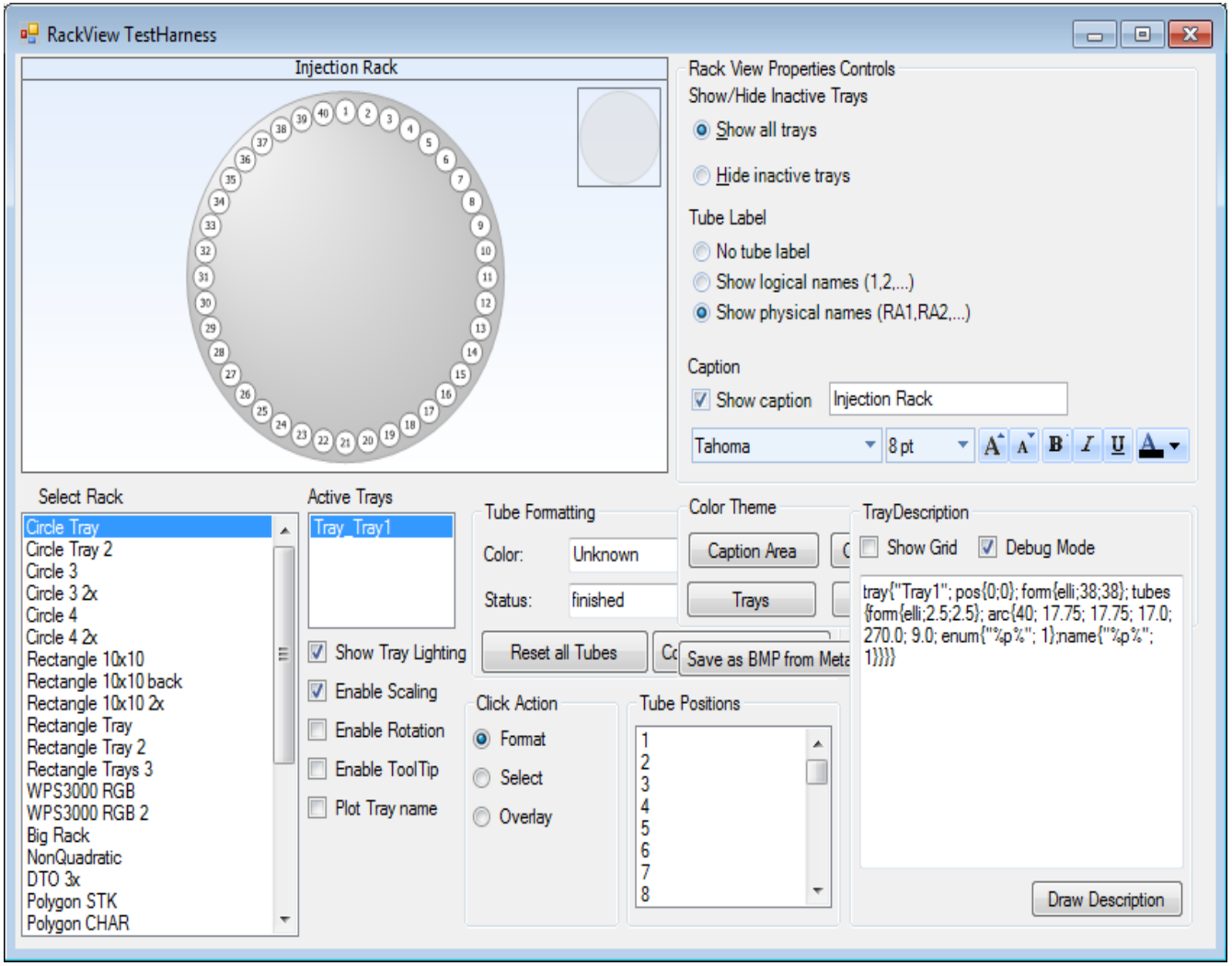
Autosampler Driver Interface - Rack

Creating a Rack View

```
IStringProperty m_RackDesciptionProperty =
m_Device.CreateStandardProperty(StandardPropertyID.TrayDescription,
m_DDK.CreateString(1000));

string rackDescription = GetRackDescription();

m_RackDesciptionProperty.Update(rackDescription);

private string GetRackDescription()
{
  IRackLayoutDescriptionBuilder builder =
        m_InjectHandler.CreateRackLayoutDescriptionBuilder();
```

Rack View Test Tool



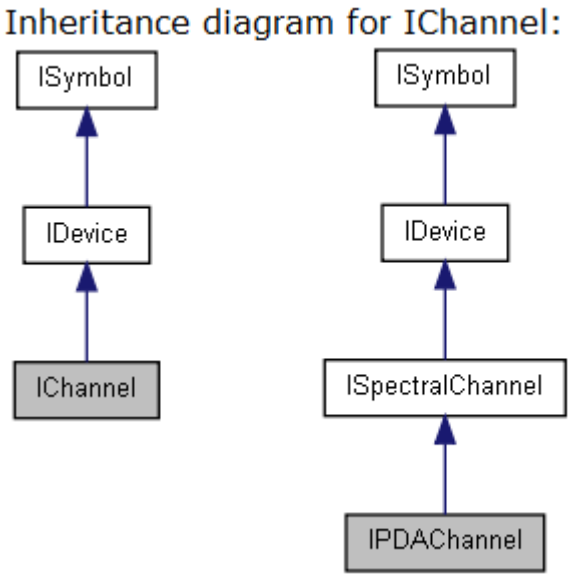


ThermoFisher SCIENTIFIC

Sending Data to Chromeleon

Channel – a device that produces data during an Injection.
The driver uses **`IDDK.CreateChannel`** to create its Channels.

- 2D data
- 3D data
- Spectrum data
- MS data




```
IChannel CreateChannel(string name, string helpText,
                                    IType signalType);
```

2D Signal Processing - IChannel

```
ITypeInt typeSignal = ddk.CreateInt(0, 1000);

typeSignal.Unit = UnitConversionEx.PhysUnitName(
                          UnitConversion.PhysUnitEnum.PhysUnit_MilliAU);
                                    // mAU - Milli-Absorbance

IChannel m_Channel = ddk.CreateChannel("Channel_1",
                                       "Detector Channel",
                                        typeSignal);
```

IChannel is a device with a predefined set of properties and commands:

m_Channel.AcquisitionOnCommand - to start data acquisition for this channel.
m_Channel.AcquisitionOffCommand - to stop data acquisition for this channel.
m_Channel.SignalProperty - has the value of the most recent data point processed.
Automatically updated by the DDK.
m_Channel.RetentionProperty - the with time of the most recent data point processed

m_Channel.AddPropertyToChannelInfo(IProperty property) - Add a channel property to the channel info of the data file.

Functions for data handling

- `m_Channel.UpdateData(int ignored, int[] data)`

  Send data points at a fixed data rate. The int parameter is kept for backward compatibility and is ignored. We recommend setting it to 0.

- `m_Channel.UpdateData(DataPoint[] data)`

  Maintained for backward compatibility only; **do not use for new drivers**. Use UpdateDataEx instead.

- `m_Channel.UpdateDataEx(DataPointEx[] data)`

  Send data points (double values) that were acquired at random times. For high data rates, we recommend using the corresponding integer function below.

- `m_Channel.UpdateDataEx(DataPointInt64[] data)`

  Send data points (integer values) that were acquired at random times.

Events used for data handling

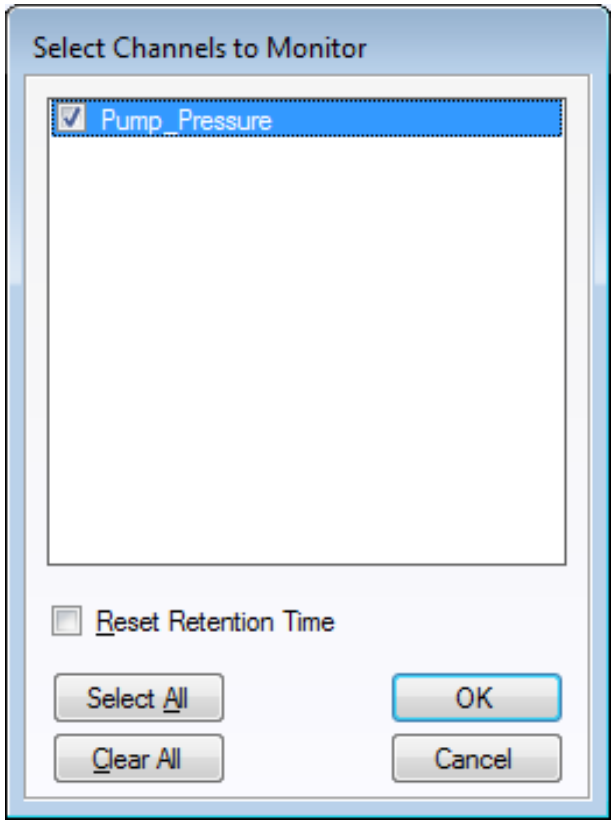
- **`m_Channel.OnDataFinished`**: The DDK fires this event when the driver has forwarded enough data (by using a series of UpdateData() calls) to the DDK. This event is typically fired at some point shortly after the Acquisition Off command has been processed. The event handler is called when the data interface had finished to saving the data and the driver may stop data acquisition.

- **`m_Channel.NoMoreData()`** A driver must call this when the hardware has stopped sending data while acquisition is on (i.e., if the data stream stops, but OnDataFinished hasn't been fired yet). Call this only if your driver detected a communication failure, a fatal error has occurred, or when the firmware signals that the data stream has ended.

2D Signal Processing

Pressure, temperature, etc. data can be monitored in ePanel

- In Monitor baseline mode
- Set a virtual channel in IM



Examples

CmDDKExamples**ChannelTest**

CmDDKExamples**ExampleLCSystem**

3D Signal Processing - IPDAChannel

3D data can be delivered to the DDK in two different ways: `IPDAChannel` and `ISpectrumWriter`

`IPDAChannel` interface is intended for PDA (photo diode array) detectors that use an array of photo diodes to measure light intensity at a large number of wavelengths in parallel (an intensity spectrum). Typically, UV transmission is calculated by either the firmware or the driver by dividing two intensity spectra recorded at acquisition start and at some time into the run.

The resulting 3D data set is UV absorption (shown in mAU in client) as a function of time and wavelength.

3D Signal Processing - IPDAChannel

The **PDA** (Photo Diode Array) Channel detectors uses an array of photo diodes to measure light intensity at a large number of wavelengths in parallel.

```
IPDAChannel CreatePDAChannel(string name, string helpText,
                             ITypeDouble wavelengthType,
                             ITypeDouble referenceWavelengthType,
                             ITypeDouble referenceBandwidthType,
                             ITypeDouble bunchWidthType,
                             int maximumNumberOfDataPoints,
                             PDAWriteMode writeMode);
```

3D Data Acquisition license

The DDK will refuse Acquisition On for `IPDAChannel` devices if the instrument controller lacks the 3D Data Acquisition license. A corresponding error message will be issued during ready check and at execution time.

Limitations

- Currently, there is no online plot available in the CM7 client for 3D data.
- `IPDAChannel` can store absorption only.

Example – CmDDKExamples**ChannelTest**

3D Signal Processing - ISpectrumWriter

`ISpectrumWriter`

This general interface can be used to store single spectra (e.g. fluorescence spectra) rather than continuous 3D absorption spectra. It is also able to store other types of continuous 3D spectra that cannot be stored by the `IPDAChannel` interface.

Create a spectrum writer with

```
IDevice.CreateSpectrumWriter()
```

The device also needs to provide suitable commands (e.g. TakeScan or even AcqOn/AcqOff) and related properties that allows the user to control when and how individual spectra should be recorded.

General workflow for `ISpectrumWriter`

1. Create the `ISpectrumWriter` object and set SpectralFieldName and DetectionPrinciple. You may use more than one `ISpectrumWriter` object for different types of spectra.
2. Use UpdateSerialNo to provide the serial number of your module (if available). This meta information will be used by all spectra that will subsequently be stored by the spectrum writer. Users can later report the value for documentation purposes.
3. Forward any instrument method properties and commands that determine spectra acquisition to your module in the usual way.
4. As you receive a single spectrum (or "scan") from the instrument,
    - set all `ISpectrumWriter` properties that describe the spectrum as appropriate.
    - set the array of DataPoints. Note that the data points need to be equidistant in the wavelength direction; and that the wavelength range is determined by WavelengthMinimum and WavelengthMaximum.
    - Save the spectrum.

3D Signal Processing - ISpectrumWriter

```
ISpectrumWriter m_SpectrumWriter = m_Device.CreateSpectrumWriter();
m_SpectrumWriter.Name = "Spectrum";
m_SpectrumWriter.Comment = "Example spectrum";
m_SpectrumWriter.DetectionPrinciple = DetectionPrinciple.Absorbance;
m_SpectrumWriter.SpectralFieldName = Id + "_Spectra";
m_SpectrumWriter.Unit = UnitConversionEx.PhysUnitName(
                        UnitConversion.PhysUnitEnum.PhysUnit_MilliAU);
m_SpectrumWriter.SignalFactor = 1;
m_SpectrumWriter.WavelengthMinimum = 2000;
m_SpectrumWriter.WavelengthMaximum = 4000;

m_SpectrumWriter.Retention = new RetentionTime(0);
m_SpectrumWriter.DataPoints = new int[200];

m_SpectrumWriter.Save();
```

3D Signal Processing - ISpectrumWriter

Contrary to the `IPDAChannel` interface, the `ISpectrumWriter` interface allows for greater flexibility. For instance, you can use it to store a few individual UV absorptions scans from a scanning VWD (Variable Wavelength Detector) to a spectrum library associated with the Injection during which the scans were produced; each scan might have a different size or resolution, and all the meta data can be different for each individual scan.

3D Signal Processing - ISpectrumWriter

You can also use the `ISpectrumWriter` interface to store non-PDA 3D data, such as a series of 3D fluorescence spectra generated at a high rate. These should probably be visualized as a 3D plot rather than as N individual scans. To request this type of behavior, set the `ISpectrumWriter` into continuous mode using `IsContinuousAcq`.

The workflow is:

- Ensure that the driver requests a 3D Data Acquisition license by calling IDevice::RegisterLicenseFeature. during IDriver.Init. (Without requesting the license, you may risk run time license failure on the first attempt to store in continuous mode if users forget to manually assign the 3D Data Acquisition license to this ICT in the Administration Console.)
- Set all meta data as stated above and set
  ```
  m_SpectrumWriter.IsContinuousAcq = true;
  ```
- As spectra are received, store them as stated above. Note that all spectra need to cover the same wavelength range, otherwise, 3D visualization might fail.
- Once the final spectrum has been received, call
  ```
  m_SpectrumWriter.NoMoreData();
  ```

3D Signal Processing - ISpectrumWriter

Example – CmDDKExamples**ChannelTest** (FixedRateChannel.cs)