SpecsLab Prodigy Remote In

VERSION 1.22
SEPTEMBER 12, 2024

Contents

1 Introduction

1.1 SpecsLab Prodigy Behavior

In remote control mode:

- Prodigy shows that it is being remotely controlled.
- Prodigy shows the data as it is acquired.
- Experiment Editor and Remote Control cannot acquire data at the same time.
- A remote acquisition can be paused or aborted within the Remote Control plugin.
- Remote Control does not interfere with running local acquisitions.
- The spectrum is recorded with a one or two-dimensional detector and is written in an m×n-array.
- During data acquisition, only one scan will be recorded. Multiple scans are taken by repeating the acquisition. Data accumulation and / or averaging as well as storage is performed by the remote client.
- Local device parameters can only be set remotely for enabled device commands of the remote experiment.

1.2 General Protocol Description

- Communication is on a request / reply basis.
- In this context, SpecsLab Prodigy acts as the server.
- Requests are sent from the client application and are answered by SpecsLab Prodigy.
- SpecsLab Prodigy only accepts a single connection.
- The protocol format is plain ASCII text via TCP/IP.
- The TCP port of the remote control server is 7010.
- Each command is acknowledged by *OK*, *OK: [...]* or *Error: <code> "message"*.
  An error condition is given by an error code and a textual description. Error codes are 16-bit positive integer values. They are not unique but indicate an error "class". Emphasis is put on the textual information.
- Each command and response are terminated by a newline character "\n".
- Token separation by <space> (ASCII 32dec).
- Message (character) strings are enclosed in double quotes: "<message>"; double quotes inside strings have to be escaped with a backslash ("\").
- Requests start with "?" followed by a request ID.
- Responses start with "!" followed by the corresponding request ID.
- Request IDs have a fixed length of 4 hexadecimal digits (e.g. 0001, AB03) where requests and responses have matching IDs.
- All command and parameter names are case sensitive.
- In the first version, no binary transfer is supported.

- Some commands which take a longer time to complete (for example, an acquisition) are performed asynchronously; a reply will be issued as a confirmation that the command will be / has been started and the actual state can be queried through other requests.
- Replies should normally be sent within one second; if a timeout occurs, the sender has to manage this depending on the command and state (resend, abort, ...).
- When disconnecting voluntarily (or when the connection is lost) the devices used during remote control are set into their respective safe states.
- Aside from the point above, no automatic error mechanism is specified with this protocol.

1.3 Request Syntax

```
?<id> Command [InParams]
```

where:

`id`	Unique request identifier (hexadecimal value, always 4 digits)
`Command`	Command name (character token, camel case, commands with spaces must be enclosed in double quotes)
`InParams`	Optional list of input parameters (“key:value”-list, space separated), specific for each command; the order of parameters is arbitrary.

EXAMPLES:

```
?0107 Connect
?0231 GetAnalyzerParameterInfo ParameterName:”Detector Voltage”
?010B DefineSpectrum StartEnergy:1.0 EndEnergy:20.0 StepWidth:1.0
[...]
?010C Disconnect
```

1.4 Response Syntax

```
!<id>       OK
```

or

```
!<id>       OK: [OutParams]
```

or

```
!<id>       Error: <Code> [Reason]
```

where:

`id`	Is the id of the corresponding request (4 digits, hexadecimal)
`OutParams`	List of output parameters (“key:value” list, space separated) or error code and textual error message
`Code`	Decimal representation of the error (see section 5).
`Reason`	Textual description of the error

EXAMPLES:

```
!0028 OK
!0028 OK: Detector Voltage:1950.0
!0198 Error: 201 Start energy should be above …
!0029 OK: ControllerStatus:running EnergyPosition:230.3
```

2 List of Commands (Requests from Client to SpecsLab Prodigy)

Every request can potentially be answered with an error reply.

2.1 Connect

Open connection to SpecsLab Prodigy.

Parameters: (None)
Response: `OK: ServerName:<Text> ProtocolVersion:<Major.Minor>`

`Text`	Arbitrary string reported from SpecsLab Prodigy
`Major`	Major number of the supported protocol version
`Minor`	Minor number of the supported protocol version

EXAMPLE:
```
?0100 Connect
!0100 OK: ServerName:“SpecsLab Prodigy 4.0” ProtocolVersion:1.2
```

2.2 Disconnect

Close connection to SpecsLab Prodigy.
When disconnecting voluntarily (or when the connection is lost) the devices used during remote control are set into their respective safe states.

Parameters: (None)
Response: `OK`

EXAMPLE:
```
?00A0 Disconnect
!00A0 OK
```

2.3 DefineSpectrumFAT

Send FAT spectrum specification for subsequent acquisition. Existing data must be cleared first.

Parameters:

`StartEnergy`	Kinetic energy of the first data point in eV
`EndEnergy`	Kinetic energy of the last data point in eV
`StepWidth`	Delta between measurement points in eV
`DwellTime`	Dwell time of the detector in seconds
`PassEnergy`	Pass energy in eV
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK`

```
EXAMPLE:
?0101 DefineSpectrumFAT StartEnergy:300.0 EndEnergy:320.0
      StepWidth:0.01 DwellTime:0.1 PassEnergy:10.0
      LensMode:"MediumArea" ScanRange:"1.5kV"
!0101 OK
```

2.4 DefineSpectrumSFAT

Send SFAT spectrum (snapshot) specification for subsequent acquisition. Existing data must be cleared first. Note: Step width and pass energy are computed automatically wrt the current detector calibration.

Parameters:

`StartEnergy`	Kinetic energy of the first data point in eV
`EndEnergy`	Kinetic energy of the last data point in eV
`Samples`	Number of acquisition samples
`DwellTime`	Dwell time of the detector in seconds
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK`

```
EXAMPLE:
?0101 DefineSpectrumSFAT StartEnergy:300.0 EndEnergy:320.0
      Samples:1 DwellTime:0.1 LensMode:"MediumArea" ScanRange:"1.5kV"
!0101 OK
```

2.5 DefineSpectrumFRR

Send FRR spectrum specification for subsequent acquisition with fixed retarding ratio. Existing data must be cleared first.

Parameters:

`StartEnergy`	Kinetic energy of the first data point in eV
`EndEnergy`	Kinetic energy of the last data point in eV
`StepWidth`	Delta between measurement points in eV
`DwellTime`	Dwell time of the detector in seconds
`RetardingRatio`	Retarding Ratio
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK`

```
EXAMPLE:
?0101 DefineSpectrumFRR StartEnergy:300.0 EndEnergy:320.0
      StepWidth:0.01 DwellTime:0.1 RetardingRatio:10.0
      LensMode:"MediumArea" ScanRange:"1.5kV"
!0101 OK
```

2.6 DefineSpectrumFE

Send FE spectrum specification for subsequent acquisition with a fixed kinetic energy. Existing data must be cleared first.

Parameters:

`KinEnergy`	Kinetic Energy in eV
`Samples`	Number of acquisition samples
`DwellTime`	Dwell time of the detector in seconds
`PassEnergy`	Pass energy in eV
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK`

```
EXAMPLE:
?0101 DefineSpectrumFE KinEnergy:300.0 Samples:5 DwellTime:0.1
      PassEnergy:10.0 LensMode:"MediumArea" ScanRange:"1.5kV"
!0101 OK
```

2.7 DefineSpectrumLVS

Send LVS spectrum specification for subsequent acquisition that scans a logical analyzer voltage. Existing data must be cleared first.

Parameters:

`Start`	Start value in logical voltage unit
`End`	End value in logical voltage unit
`StepWidth`	Step size between two logical voltages
`KinEnergy`	Kinetic Energy in eV
`DwellTime`	Dwell time of the detector in seconds
`PassEnergy`	Pass energy in eV
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)
`ScanVariable`	Scan variable (as string)

Response: `OK`

EXAMPLE:

```
?0101 DefineSpectrumLVS Start:-1 End:1 StepWidth:0.1 KinEnergy:280
      DwellTime:0.1 PassEnergy:10.0 LensMode:"MediumArea"
      ScanRange:"1.5kV" ScanVariable:"Focus Displacement 1 [nu]"
!0101 OK
```

2.8 CheckSpectrumFAT

Validate FAT spectrum specification without setting it for subsequent acquisition. The existing acquisition status will be kept.

Parameters:

`StartEnergy`	Kinetic energy of the first data point in eV
`EndEnergy`	Kinetic energy of the last data point in eV
`StepWidth`	Delta between measurement points in eV
`DwellTime`	Dwell time of the detector in seconds
`PassEnergy`	Pass energy in eV
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK: [OutParams]`

`OutParams` “`key:value`” list of the actual parameter values of the spectrum command (potentially modified during the validation).

EXAMPLE:

```
?0103 CheckSpectrumFAT StartEnergy:300.0 EndEnergy:320.0
      StepWidth:0.01 DwellTime:0.1 PassEnergy:10.0
      LensMode:"MediumArea" ScanRange:"1.5kV"
!0103 OK: StartEnergy:300.0 EndEnergy:320.0 StepWidth:0.01
      Samples:2001 DwellTime:0.1 PassEnergy:10.0
      LensMode:"MediumArea" ScanRange:"1.5kV"
```

2.9 CheckSpectrumSFAT

Validate SFAT spectrum (snapshot) specification without setting it for subsequent acquisition. The existing acquisition status will be kept. Note: Step width and pass energy are computed automatically wrt the current detector calibration.

Parameters:

`StartEnergy`	Kinetic energy of the first data point in eV
`EndEnergy`	Kinetic energy of the last data point in eV
`Samples`	Number of acquisition samples
`DwellTime`	Dwell time of the detector in seconds
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK: [OutParams]`

`OutParams`	“`key:value`” list of the actual parameter values of the spectrum command (potentially modified during the validation).

EXAMPLE:

```
?0103 CheckSpectrumSFAT StartEnergy:300.0 EndEnergy:320.0
      Samples:1 DwellTime:0.1 LensMode:"MediumArea" ScanRange:"1.5kV"
!0103 OK: StartEnergy:300.0 EndEnergy:320.0 StepWidth:2.5
      Samples:1 DwellTime:0.1 PassEnergy:96.1099
      LensMode:"MediumArea" ScanRange:"1.5kV"
```

2.10 CheckSpectrumFRR

Validate FRR spectrum specification without setting it for subsequent acquisition. The existing acquisition status will be kept.

Parameters:

`StartEnergy`	Kinetic energy of the first data point in eV
`EndEnergy`	Kinetic energy of the last data point in eV
`StepWidth`	Delta between measurement points in eV
`DwellTime`	Dwell time of the detector in seconds
`RetardingRatio`	Retarding Ratio
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK: [OutParams]`

`OutParams`	“`key:value`” list of the actual parameter values of the spectrum command (potentially modified during the validation).

EXAMPLE:

```
?0103 CheckSpectrumFRR StartEnergy:300.0 EndEnergy:320.0
      StepWidth:0.01 DwellTime:0.1 RetardingRatio:10.0
      LensMode:"MediumArea" ScanRange:"1.5kV"
!0103 OK: StartEnergy:300.0 EndEnergy:320.0 StepWidth:0.01
      Samples:2001 DwellTime:0.1 PassEnergy:30.0
      LensMode:"MediumArea" ScanRange:"1.5kV"
```

2.11 CheckSpectrumFE

Validate FE spectrum specification without setting it for subsequent acquisition. The existing acquisition status will be kept.

Parameters:

`KinEnergy`	Kinetic Energy in eV
`Samples`	Number of acquisition samples
`DwellTime`	Dwell time of the detector in seconds
`PassEnergy`	Pass energy in eV
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)

Response: `OK: [OutParams]`

`OutParams`	`“key:value”` list of the actual parameter values of the spectrum command (potentially modified during the validation).

EXAMPLE:

```
?0103 CheckSpectrumFE KinEnergy:300.0 Samples:5 DwellTime:0.1
      PassEnergy:10.0 LensMode:"MediumArea" ScanRange:"1.5kV"
!0103 OK: StartEnergy:0 EndEnergy:4 StepWidth:1
      Samples:5 DwellTime:0.1 PassEnergy:10
      LensMode:"MediumArea" ScanRange:"1.5kV"
```

2.12 CheckSpectrumLVS

Validate LVS spectrum specification without setting it for subsequent acquisition. The existing acquisition status will be kept.

Parameters:

`Start`	Start value in logical voltage unit
`End`	End value in logical voltage unit
`StepWidth`	Step size between two logical voltages
`KinEnergy`	Kinetic Energy in eV
`DwellTime`	Dwell time of the detector in seconds
`PassEnergy`	Pass energy in eV
`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)
`ScanVariable`	Scan variable (as string)

Response: `OK: [OutParams]`

`OutParams`	`“key:value”` list of the actual parameter values of the spectrum command (potentially modified during the validation).

EXAMPLE:

```
?0101 CheckSpectrumLVS Start:10 End:20 StepWidth:1 KinEnergy:280
      DwellTime:0.1 PassEnergy:10.0 LensMode:"MediumArea"
      ScanRange:"1.5kV" ScanVariable:"Focus Displacement 1 [nu]"
!0103 OK: Start:10 End:20 StepWidth:1 KinEnergy:280
      DwellTime:0.1 PassEnergy:10.0 LensMode:"MediumArea"
      ScanRange:"1.5kV" ScanVariable:"Focus Displacement 1 [nu]"
```

2.13 ValidateSpectrum

Validate parameters defined by previous `DefineSpectrum<Type>` command. Existing data must be cleared first.

Parameters: (None)
Response: `OK: [OutParams]`

`OutParams`	`“key:value”` list of the actual parameter values of the spectrum command (potentially modified during the validation).

```
EXAMPLE:
?0102 ValidateSpectrum
!0102 OK: StartEnergy:300.0 EndEnergy:320.0 StepWidth:0.01
      Samples:2001 DwellTime:0.1 PassEnergy:10.0
      LensMode:"MediumArea" ScanRange:"1.5kV"
```

2.14 Start

Start data acquisition. Spectrum must have been validated first. An acquired spectrum remains valid when it is cleared.

Parameters:

`SetSafeStateAfter`	Specifies whether the analyzer should be set into the safe state after the scan or not (Boolean value, as string). If set to “false” the detector voltage is **not** ramped down after the scan and prone to damage by other sources (like ion sources). The parameter is optional. If not specified, the analyzer is set into its safe state (as if set to “true”).

Response: `OK`

```
EXAMPLE:
?0102 Start
!0102 OK

?0102 Start SetSafeStateAfter:"false"
!0102 OK
```

2.15 Pause

Pause data acquisition.

Parameters: (None)
Response: OK

```
EXAMPLE:
?0102 Pause
!0102 OK
```

2.16 Resume

Resume a paused data acquisition.

Parameters: (None)
Response: OK

```
EXAMPLE:
?0102 Resume
!0102 OK
```

2.17 Abort

Abort a running or paused data acquisition.

Parameters: (None)
Response: OK

```
EXAMPLE:
?0102 Abort
!0102 OK
```

2.18 GetAcquisitionStatus

Reports information about the status and the progress of the acquisition.

Parameters: (None)

Response: `OK: ControllerState:<ContState> NumberOfAcquiredPoints:<NumPts> [optional: Message:<Text> Details:<Text>]`

`ContState:`	`idle`	No spectrum is specified or a spectrum is not validated
	`validated`	Spectrum has successfully been validated
	`running`	Acquisition is running
	`paused`	Acquisition has been paused
	`finished`	Acquisition is finished (or has been aborted) and spectrum has not been cleared
	`aborted`	Acquisition has been aborted
	`error`	An error occurred

`NumberOfAcquiredPoints`	positive integer value
`Message`	Error Message
`Details`	Error Details

EXAMPLES:

```
?0102 GetAcquisitionStatus
!0102 OK: ControllerState:idle

?0103 GetAcquisitionStatus
!0103 OK: ControllerState:validated

?0104 GetAcquisitionStatus
!0104 OK: ControllerState:running NumberOfAcquiredPoints:12

?0105 GetAcquisitionStatus
!0105 OK: ControllerState:paused NumberOfAcquiredPoints:75

?0106 GetAcquisitionStatus
!0106 OK: ControllerState:finished NumberOfAcquiredPoints:92
```

2.19 GetAcquisitionData

Request a slice of data from the acquisition buffer. The buffer is not modified through this reading (non-destructive). Reading from parts of the buffer which have not been acquired results in an error.

Parameters:

`FromIndex`	Index of first sample to be reported
`ToIndex`	Index of last sample to be reported

Response: `OK: Data:[Values]`

`Values`

(S)FAT, FRR, FE: List of double values which have to be interpreted as a two-dimensional data set (non-energy channels) × (samples) of the form

[s_1i, ..., s_1j, s_2i, ..., s_2j, ..., s_Mi, ..., s_Mj]

where M equals the number of non-energy channels, i = FromIndex and j = ToIndex.

LVS: List of double values which have to be interpreted as a three-dimensional data set (samples) × (non-energy channels) × (energy channels) of the form

[s_i11, ..., s_i1N, s_i21, ..., s_i2N, ..., s_iM1, ..., s_iMN, ...,
s_j11, ..., s_j1N, s_j21, ..., s_j2N, ..., s_jM1, ..., s_jMN]

where M equals the number of non-energy channels, N equals the number of energy channels, i = FromIndex and j = ToIndex. Note: *samples* equals the number of scan variable steps.

KREIOS MM k-space data is provided in the same structure as LVS where k_x is mapped to the energy channels and k_y to non-energy channels.

EXAMPLES:

```
?0102 GetAcquisitionData FromIndex:2 ToIndex:4
!0102 OK: Data:[247599,246218,240558,233324,230841,230169, …]
```

2.20 ClearSpectrum

If controller is in state finished, the command clears the internal spectrum buffer and sets the controller state to idle. During an acquisition an error is reported. A cleared spectrum remains valid until a new definition is send.

Parameters: (None)
Response: `OK`

```
EXAMPLE:
?0102 ClearSpectrum
!0102 OK
```

2.21 GetAllAnalyzerParameterNames

Request all analyzer device parameter names.

Parameters: (None)
Response: `OK: ParameterNames:[Names]`

`Names`	List of parameter names

```
EXAMPLE:
?0231 GetAllAnalyzerParameterNames
!0231 OK: ParameterNames:["Detector Voltage","Kinetic Energy Base",…]
```

2.22 GetAnalyzerParameterInfo

Request information about a single analyzer parameter.

Parameters:

`ParameterName`	Name of the parameter whose information is queried

Response: `OK: Type:<Type> ValueType:<ValueType> Unit:<Unit> [optional: Min:<Min> Max:<Max> Values:[…]]`

`Type`	LogicalVoltage or Setting
`ValueType`	bool, double, integer, string
`Unit`	parameter unit if available, otherwise empty
`Min`	minimum value if available, otherwise skipped
`Max`	maximum value if available, otherwise skipped
`Values`	enumeration of valid values if available, otherwise skipped

EXAMPLE:

```
?0231 GetAnalyzerParameterInfo ParameterName:"Detector Voltage"
!0231 OK: Type:LogicalVoltage ValueType:double Unit:"V"

?0232 GetAnalyzerParameterInfo ParameterName:"Analyzer Standby Delay"
!0232 OK: Type:Setting ValueType:double Unit:"s"

?0233 GetAnalyzerParameterInfo ParameterName:"Skip Delay Up/Down"
!0233 OK: Type:Setting ValueType:bool Unit:””
```

2.23 GetAnalyzerVisibleName

Request the analyzer device visible name.

Parameters: (None)
Response: `OK: AnalyzerVisibleName:Name`

`Name`	The analyzer device visible name.

EXAMPLE:

```
?0231 GetAnalyzerVisibleName
!0231 OK: AnalyzerVisibleName:”Phoibos HSA3500 150 R7 NAP”
```

2.24 GetAnalyzerParameterValue

Request the value of a single analyzer parameter.
Note that this is not the current HSA voltage but represents the value used for an acquisition which can be defined through `SetAnalyzerParameterValue`.

Parameters:

`ParameterName`	Name of the parameter which is queried

Response: `OK: <ParameterName>:<ParameterValue>`

`ParameterName`	Name of the reported parameter setting
`ParameterValue`	Value of the queried parameter

```
EXAMPLE:
?0231 GetAnalyzerParameterValue ParameterName:"Detector Voltage"
!0231 OK: Name:”Detector Voltage” Value:1850.0

?0232 GetAnalyzerParameterValue ParameterName:"Skip Delay Up/Down"
!0232 OK: Name:”Skip Delay Up/Down” Value:”true”
```

2.25 SetAnalyzerParameterValue

Sets the value of a single analyzer parameter.
Parameter voltages and (for 2D detectors) the number of energy channels and non-energy channels can be set.
Parameters can only be set if no acquisition is running. After changing the number of channels, the spectrum has to be validated again.

Parameters:

`ParameterName`	Name of the analyzer parameter
`Value`	Value (unit is depending on the specific parameter)

Response: `OK`

```
EXAMPLE:
?0231 SetAnalyzerParameterValue ParameterName:"Kinetic Energy Base"
      Value:10.0
!0231 OK
```

2.26 SetAnalyzerParameterValueDirectly

Sets one or more logical voltages / currents directly without an acquisition. Returns when the parameter has been set.
Parameters can only be set if no acquisition is running. Note that for all parameters not set explicitly, the values from the analyzer are used.
This has no effect to the voltages / currents defined for an acquisition. This can only be done through `SetAnalyzerParameterValue`.

Parameters:

`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)
`Polarity`	Polarity to use, either “negative” or “positive” (as string)

One or more Parameter as `ParameterName:Value`

where ParameterName is the name of the analyzer parameter and value is the corresponding value to set. All Parameters of Type LogicalVoltage that can be retrieved by GetAllAnalyzerParameter are valid. Additionally “Kinetic Energy” and “Pass Energy” can be used.

Response: `OK`

EXAMPLE:

```
?0232 SetAnalyzerParameterValueDirectly LensMode:"MediumArea"
      ScanRange:"1.5kV" Polarity:"negative" "Kinetic Energy":120
      "Pass Energy":20
!0232 OK
```

2.27 ValidateAnalyzerParameterValueDirectly

Validates one or more logical voltages / currents as preparation for SetAnalyzerParameterValueDirectly. Returns when all parameters are valid to be set. Note that for all parameters not set explicitly, the values from the analyzer are used.

Parameters:

`LensMode`	Lens mode (as string)
`ScanRange`	HSA voltage range for scanning (as string)
`Polarity`	Polarity to use, either “negative” or “positive” (as string)

One or more Parameter as `ParameterName:Value`

where ParameterName is the name of the analyzer parameter and value is the corresponding value to be validated. All Parameters of Type LogicalVoltage that can be retrieved by GetAllAnalyzerParameter are valid. Additionally “Kinetic Energy” and “Pass Energy” can be used.

Response: `OK`

EXAMPLE:

```
?0232 ValidateAnalyzerParameterValueDirectly LensMode:"MediumArea"
      ScanRange:"1.5kV" Polarity:"negative" "Kinetic Energy":120
      "Pass Energy":20
!0232 OK
```

2.28 GetSpectrumParameterInfo

Request information about a single spectrum parameter.

Parameters:

`ParameterName`	Name of the parameter whose information is queried (cmp. DefineSpectrum)

Response: `OK: ValueType:<ValueType> Unit:<Unit> [optional: Min:<Min> Max:<Max> Values:[…]]`

`ValueType`	bool, double, integer, string
`Unit`	parameter unit if available, otherwise empty
`Min`	minimum value if available, otherwise skipped
`Max`	maximum value if available, otherwise skipped
`Values`	enumeration of valid values if available, otherwise skipped

EXAMPLE:
```
?0231 GetSpectrumParameterInfo ParameterName:"LensMode"
!0231 OK: ValueType:string Unit:""
Values:["HighMagnification","HighPointTransmission","LargeArea","Medi
umArea","MediumMagnification","MediumPointTransmission"]
```

2.29 GetSpectrumDataInfo

Request information about a single spectrum data parameter.

Parameters:

`ParameterName`	Name of the parameter whose information is queried (currently only “OrdinateRange” and “AbscissaRange” are supported)

Response: `OK: ValueType:<ValueType> Unit:<Unit> [optional: Min:<Min> Max:<Max> Values:[…]]`

`ValueType`	bool, double, integer, string
`Unit`	parameter unit if available, otherwise empty
`Min`	minimum value if available, otherwise skipped
`Max`	maximum value if available, otherwise skipped
`Values`	enumeration of valid values if available, otherwise skipped

EXAMPLE:

```
?0231 GetSpectrumDataInfo ParameterName:"OrdinateRange"
!0231 OK: ValueType:double Unit:"deg" Min:-0.571875 Max:1.77187
```

2.30 GetAllDeviceCommands

Request list of available device commands. It returns all devices resp. according commands which are defined within the remote experiment.

Parameters:	(None)
Response:	`OK: DeviceCommands:[Names]`
`Names`	List of device command names. A device command name consists of the system unique device name and the device command specified in the remote experiment. The format is: `"<DeviceName>.<CommandName>"`

EXAMPLE:

```
?0231 GetAllDeviceCommands
!0231 OK: DeviceCommands:[ "XRC125MF.Activate Preset","Phoibos1D.Set
Parameters","FOCUSMagneticPulse.Operate",…]
```

2.31 GetAllDeviceParameterNames

Request all device parameter names of a certain device command which has to be defined within the remote experiment. Parameters which can be selected within the device command have to be enabled for remote access.

Parameters:

`DeviceCommand`	Name of the device command whose parameters are queried (cmp. GetAllDeviceCommands)
Response:	`OK: ParameterNames:[Names]`
`Names`	List of device parameter names

EXAMPLE:

```
?0231 GetAllDeviceParameterNames
DeviceCommand:"FOCUSMagneticPulse.Operate"
!0231 OK: ParameterNames:["ChargeVoltage","Coil","NegativePolarity"]
```

2.32 GetDeviceParameterInfo

Request information about a single device parameter.

Parameters:

`ParameterName`	Name of the device parameter whose information is queried
`DeviceCommand`	Name of the according device command

Response: `OK: Type:<Type> ValueType:<ValueType> Unit:<Unit> [optional: Values:[…]]`

`Type`	DeviceParameter
`ValueType`	bool, double, integer, string
`Unit`	parameter unit if available, otherwise empty
`Values`	enumeration of valid values if available, otherwise skipped

EXAMPLE:

```
?0231 GetDeviceParameterInfo ParameterName:"ChargeVoltage"
      DeviceCommand:"FOCUSMagneticPulse.Operate"
!0231 OK: Type:DeviceParameter ValueType:double Unit:"V"


?0232 GetDeviceParameterInfo ParameterName:"NegativePolarity"
      DeviceCommand:"FOCUSMagneticPulse.Operate"
!0232 OK: Type:DeviceParameter ValueType:string Unit:""
Values:["ON","OFF"]
```

2.33 GetDeviceParameterValue

Request the value of a single device parameter.

Parameters:

`ParameterName`	Name of the device parameter which is queried
`DeviceCommand`	Name of the according device command

Response: `OK: Name:<ParameterName> Value:<ParameterValue>`

`ParameterName`	Name of the reported device parameter
`ParameterValue`	Value of the queried parameter

```
EXAMPLE:
?0231 GetDeviceParameterValue ParameterName:"ChargeVoltage"
      DeviceCommand:"FOCUSMagneticPulse.Operate"
!0231 OK: Name:"ChargeVoltage" Value:0

?0232 GetDeviceParameterValue ParameterName:"NegativePolarity"
      DeviceCommand:"FOCUSMagneticPulse.Operate"
!0232 OK: Name:"NegativePolarity" Value:"ON"
```

2.34 SetDeviceParameterValue

Change the value of a device parameter.
Parameters can only be set if no acquisition is running.

Parameters:

`ParameterName`	Name of the device parameter which is queried
`DeviceCommand`	Name of the according device command
`Value`	Value (depending on the specific parameter)

Response: `OK`

```
EXAMPLE:
?0231 SetDeviceParameterValue ParameterName:"ChargeVoltage"
      DeviceCommand:"FOCUSMagneticPulse.Operate" Value:1.0
!0231 OK

?0232 SetDeviceParameterValue ParameterName:"NegativePolarity"
      DeviceCommand:"FOCUSMagneticPulse.Operate" Value:"OFF"
!0232 OK
```

2.35 DisconnectAnalyzer

Request to disconnect analyzer.

Parameters: (None)
Response: `OK`

```
EXAMPLE:
?0231 DisconnectAnalyzer
!0231 OK
```

2.36 SetSafeState

Request to set all devices into safe state.
This will actively wait for the devices to reach their targeted state. Depending on the current state of analyzer etc. this may take a while. An error (215) is returned if any of the devices did not reach their safe state within one minute.

Parameters: (None)
Response: `OK`

```
EXAMPLE:
?0232 SetSafeState
!0232 OK
```

2.37 CreateDirectDeviceCommand

Creates an experiment item for operating on Devices by loading an appropriate template. The Devices item may not contain child items and there can only be one Devices item, so any such previously created item will be replaced.

Returns the list of commands which are defined within the Devices item.

Parameters:

`Template`	Prodigy template to be loaded
`TemplateGroup`	Name of the template group (optional parameter)

Response: `OK: DeviceCommands:[< Names>]`

`Names`	List of device command names (see GetAllDeviceCommands for notation)

```
EXAMPLE:
?0233 CreateDirectDeviceCommand Template:”Gas Flow”
!0233 OK: DeviceCommands:[“BrooksGF040.Operate”]
```

2.38 GetDirectDeviceCommandInfo

Request information about a device command from the Devices item created via CreateDirectDeviceCommand.

Parameters:

`DeviceCommand`	Name of the according device command

Response: `OK: Type:<Type> Name:<Name> ParameterNames:[<Names>]`

`Type`	Device type
`Name`	Device name
`ParameterNames`	List of device parameter names

```
EXAMPLE:
?0234 GetDirectDeviceCommandInfo DeviceCommand:" BrooksGF040.Operate"
!0234 OK: Type:”Brooks GF 040” Name:”Brooks Mass Flow Controller”
      ParameterNames:[“mass_flow”]
```

2.39 GetDirectDeviceParameterInfo

Request information about a parameter of a device command created via CreateDirectDeviceCommand.

Parameters:

`DeviceCommand`	Name of the according device command
`ParameterName`	Name of the device parameter whose information is queried

Response: `OK: Type:<Type> ValueType:<ValueType> Unit:<Unit> [optional: Values:[…]]`

`Type`	DeviceParameter
`ValueType`	bool, double, integer, string
`Unit`	parameter unit if available, otherwise empty
`Values`	enumeration of valid values if available, otherwise skipped

EXAMPLE:

```
?0235 GetDirectDeviceParameterInfo
      DeviceCommand:"BrooksMFC040.Operate"
      ParameterName:"mass_flow"
!0235 OK: Type:DeviceParameter ValueType:double Unit:"ml/min"
```

2.40 GetDirectDeviceParameterValue

Request the parameter value of a device command created via CreateDirectDeviceCommand.

Parameters:

`DeviceCommand`	Name of the according device command
`ParameterName`	Name of the device parameter which is queried

Response: `OK: Name:<ParameterName> Value:<ParameterValue>`

`ParameterName`	Name of the reported device parameter
`ParameterValue`	Value of the queried parameter

```
EXAMPLE:
?0236 GetDirectDeviceParameterValue
      DeviceCommand:"BrooksGF040.Operate"
      ParameterName:"mass_flow"
!0236 OK: Name:"mass_flow" Value:0
```

2.41 SetDirectDeviceParameterValue

Change the parameter value of a device command created via CreateDirectDeviceCommand.
Parameters can only be set if the Devices item is not running.

Parameters:

`ParameterName`	Name of the device parameter which is queried
`DeviceCommand`	Name of the according device command
`Value`	Value (depending on the specific parameter)

Response: `OK`

```
EXAMPLE:
?0237 SetDirectDeviceParameterValue
      DeviceCommand:"BrooksGF040.Operate"
      ParameterName:"mass_flow" Value:250.0
!0237 OK
```

2.42 ExecuteDirectDeviceCommand

Run the Devices item created by CreateDirectDeviceCommand.

Parameters:

`SetSafeStateAfter`	Specifies whether the used devices should be set into the safe state after the execution was finished (Boolean value, as string). The parameter is optional. If not specified, the devices are set into their corresponding safe state (as if set to “true”).

Response: `OK`

```
EXAMPLE:
?0238 ExecuteDirectDeviceCommand
!0239 OK
```

2.43 GetAllDevices

Request a list of available devices. It returns all devices which are defined in the system configuration.

Parameters: (None)
Response: `OK: Devices:[Names]`

`Names`	List of system unique device names

```
EXAMPLE:
?0231 GetAllDevices
!0231 OK: Devices:[ "XRC 125 MF","Phoibos 1D"]
```

2.44 GetDeviceInfo

Request device information of a certain device in the system configuration.

Parameters:

`Device`	Name of the device whose information is queried (cmp. GetAllDevices)

Response:
```
OK: Type:<DeviceType>
VisibleName:<DeviceVisibleName>
LiveParameterNames:[Names]
```

`Type`	Device type
`VisibleName`	Device visible name
`LiveParameterNames`	List of device parameter names, may be empty if no parameters exist

```
EXAMPLE:
?0231 GetDeviceInfo Device:"Analyzer 1D"
!0231 OK: Type:"Phoibos1D" VisibleName:"Analyzer"
LiveParameterNames:["Kinetic Energy (Target)","Pass Energy
(Target)","Detector Voltage (Target)","Count Rate"]
```

2.45 GetLiveParameterInfo

Requesting information about a device parameter.

Parameters:

`Device`	Name of the corresponding device (cmp. GetAllDevices)
`Parameter`	Name of the device parameter whose information is being queried (cmp. GetDeviceInfo)

Response: `OK: ValueType:<ValueType> Unit:<Unit> [optional: Values:[…]]`

`ValueType`	bool, double, integer, string
`Unit`	parameter unit if available, otherwise empty
`Values`	enumeration of valid values if available, otherwise skipped

EXAMPLE:

```
?0235 GetLiveParameterInfo Device:"XRC 125 MF" Parameter:"Voltage"
!0235 OK: ValueType:double Unit:"V"
```

2.46 GetLiveParameterValue

Requesting the current value of a device parameter.

Parameters:

`Device`	Name of the corresponding device (cmp. GetAllDevices)
`Parameter`	Name of the device parameter whose information is being queried (cmp. GetDeviceInfo)

Response: `OK: Connectivity:<Connectivity> Value:<Value>`

`Connectivity`	`Online`	device is connected and the value can be retrieved
	`Offline`	device is not connected.
	`Unresponsive`	device connection is lost
	For both of these conditions the last available value will be displayed	
`Value`	Value of the requested parameter	

EXAMPLE:

```
?0236 GetLiveParameterValue Device:"XRC 125 MF" Parameter:"Voltage"
!0236 OK: Connectivity:Online Value:1.53846153
```

3 Remote Session Examples

The TCP port of the remote control server is 7010.

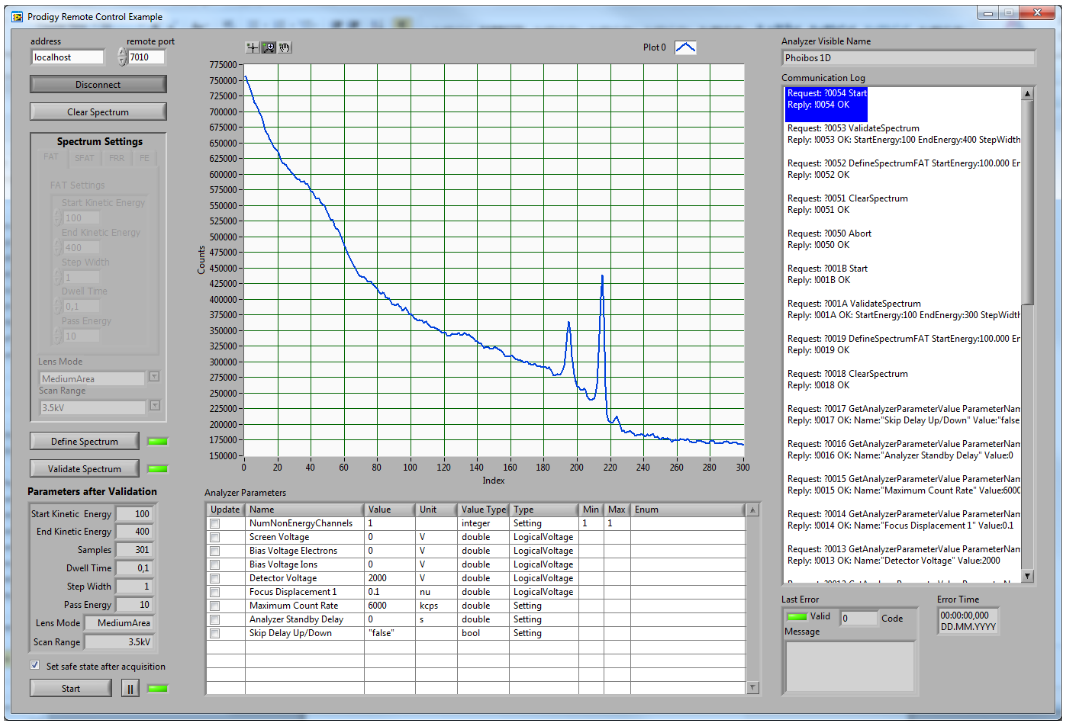
3.1 Protocol Example

```
?0001 Connect
!0001 OK: ServerName:"SpecsLab Prodigy 4.8-r44312"
ProtocolVersion:1.4

?0002 GetAllAnalyzerParameterNames
!0002 OK: ParameterNames:["NumEnergyChannels","Screen Voltage","Bias
Voltage Electrons","Bias Voltage Ions","Detector Voltage","Focus
Displacement 1","Maximum Count Rate [kcps]","Analyzer Standby Delay
[s]","Skip Delay Up/Down"]

?0003 GetAnalyzerParameterInfo ParameterName:"Screen Voltage"
!0003 OK: Type:LogicalVoltage ValueType:double Unit:""

?0004 GetAnalyzerParameterValue ParameterName:"Screen Voltage"
!0004 OK: Name:"Screen Voltage" Value:0

?0005 ValidateSpectrum
!0005 Error: 202 Remote Control: Validation failed.
No lens mode specified.
Please select a valid lens mode for spectrum 'Remote Control'.

?0006 DefineSpectrumFAT StartEnergy:300.0 EndEnergy:1500.0
StepWidth:1 DwellTime:0.1 PassEnergy:10.0 LensMode:"MediumArea"
ScanRange:"1.5kV"
!0006 OK

?0007 ValidateSpectrum
!0007 OK: StartEnergy:300 EndEnergy:1500 StepWidth:1 DwellTime:0.1
PassEnergy:10 LensMode:"MediumArea" ScanRange:"1.5kV"

?0008 Start
!0008 OK

?0009 Pause
```

```
!0009 OK

?0010 Resume
!0010 OK

?0011 GetAcquisitionStatus
!0011 OK: ControllerState:finished NumberOfAcquiredPoints:1194

?0012 GetAcquisitionData FromIndex:0 ToIndex:8
!0012 OK: Data:[247461,243729,239662,235772,232407,231056,232737,
238976,257519]

?0013 Disconnect
```

3.2 LabVIEW Example

There is a LabVIEW example available in the Prodigy program folder (e.g. *C:\Program Files (x86)\SPECS\SpecsLab Prodigy\Programming Examples\LabVIEW\ProdigyRemoteControl*). The LabVIEW-Project includes a library with most supported functions of the remote protocol. Backup and extend the project or use a copy of the library in an own project.



Parameters after Validation

Start Kinetic Energy	100
End Kinetic Energy	400
Samples	301
Dwell Time	0,1
Step Width	1
Pass Energy	10
Lens Mode	MediumArea
Scan Range	3.5kV

Analyzer Parameters

Update	Name	Value	Unit	Value Type	Type	Min	Max	Enum
☐	NumNonEnergyChannels	1		integer	Setting	1	1	
☐	Screen Voltage	0	V	double	LogicalVoltage			
☐	Bias Voltage Electrons	0	V	double	LogicalVoltage			
☐	Bias Voltage Ions	0	V	double	LogicalVoltage			
☐	Detector Voltage	2000	V	double	LogicalVoltage			
☐	Focus Displacement 1	0.1	nu	double	LogicalVoltage			
☐	Maximum Count Rate	6000	kcps	double	Setting			
☐	Analyzer Standby Delay	0	s	double	Setting			
☐	Skip Delay Up/Down	"false"		bool	Setting			

Analyzer Visible Name: Phoibos 1D

Communication Log

```
Request: ?0054 Start
Reply: !0054 OK

Request: ?0053 ValidateSpectrum
Reply: !0053 OK: StartEnergy:100 EndEnergy:400 StepWidth

Request: ?0052 DefineSpectrumFAT StartEnergy:100.000 Er
Reply: !0052 OK

Request: ?0051 ClearSpectrum
Reply: !0051 OK

Request: ?0050 Abort
Reply: !0050 OK

Request: ?001B Start
Reply: !001B OK

Request: ?001A ValidateSpectrum
Reply: !001A OK: StartEnergy:100 EndEnergy:300 StepWidtl

Request: ?0019 DefineSpectrumFAT StartEnergy:100.000 Er
Reply: !0019 OK

Request: ?0018 ClearSpectrum
Reply: !0018 OK

Request: ?0017 GetAnalyzerParameterValue ParameterNan
Reply: !0017 OK: Name:"Skip Delay Up/Down" Value:"false

Request: ?0016 GetAnalyzerParameterValue ParameterNan
Reply: !0016 OK: Name:"Analyzer Standby Delay" Value:0

Request: ?0015 GetAnalyzerParameterValue ParameterNan
Reply: !0015 OK: Name:"Maximum Count Rate" Value:600C

Request: ?0014 GetAnalyzerParameterValue ParameterNan
Reply: !0014 OK: Name:"Focus Displacement 1" Value:0.1

Request: ?0013 GetAnalyzerParameterValue ParameterNan
Reply: !0013 OK: Name:"Detector Voltage" Value:2000
```

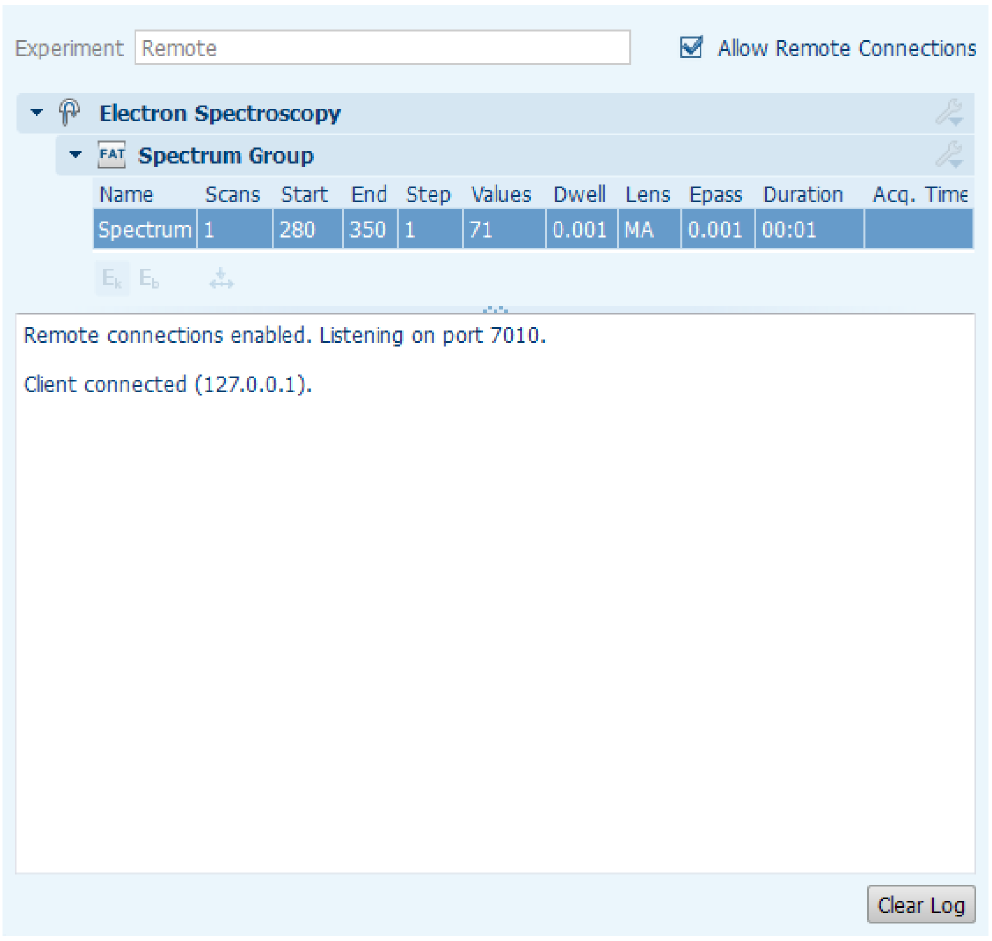
Last Error: Valid 🟩, Code 0, Message

Error Time: 00:00:00,000 DD.MM.YYYY

4 Plug-in Interface

In order to establish a remote connection:
1. Open the Remote Control View.
2. Select an experiment template from the Schedule-Selector.
3. Enable remote connections.
4. Communicate via TCP (see above).

During the remote session you can see the requests and replies in the lower part of the window.



Experiment: Remote

Allow Remote Connections ✓

Electron Spectroscopy

Spectrum Group

Name	Scans	Start	End	Step	Values	Dwell	Lens	Epass	Duration	Acq. Time
Spectrum	1	280	350	1	71	0.001	MA	0.001	00:01	

```
Remote connections enabled. Listening on port 7010.

Client connected (127.0.0.1).
```

NOTE:
A new configuration can be specified with the Experiment Editor plug-in and must consist of a single spectrum definition in a single Electron Spectroscopy element. It is important that the configuration:

1. Contains a valid analyzer and source command, and
2. Specifies a correct detector calibration plus analyzer slits.

All other parameters will be set via remote requests.

The default folder for configurations is <SPECS Settings Folder>\RemoteControl. An existing default configuration (Default.slt) will be loaded automatically when the plugin is opened.

5 List of Error Codes

Remote Control errors are primarily categorized by their layer:

- Connection Errors (Range 1 .. 99)
- Protocol Errors (Range 101 .. 199)
- Logical and Execution Errors (Range 201 .. 299)

The following gives an overview of the categories and the corresponding error codes.
A more detailed cause will be given in the textual error description of the response message (see section 1.4).

5.1 Connection Errors

These errors signal a failure at the message-passing level between client and server or a malformed incoming message.

Error	Reason
1	No server to connect to
2	Another client is already connected
3	Client is not connected
4	Malformed message

5.2 Protocol Errors

Protocol errors happen when a request has been forwarded to the server but is formatted incorrectly or the command arguments do not match the command's specification.

Error	Reason
101	Unknown command
102	Unknown error
103	Invalid argument sequence
104	Missing argument
105	Unknown argument
106	Invalid argument type
107	Invalid argument value

5.3 Logical and Execution Errors

Logical and execution errors occur when the command itself is syntactically correct but cannot be executed in the specified context. They may also arise on any other failure when the command is executed (e.g. a device error occurs).

Error	Reason
201	Failed to set the spectrum parameters
202	Validation error
203	Failed to start acquisition
204	Failed to clear spectrum
205	Failed to fetch parameter info
206	Unknown parameter
207	No data available
208	Invalid range
209	Currently acquiring spectrum
210	Spectrum contains data
211	Spectrum not validated
212	No running acquisition
213	Failed to disconnect analyzer
214	Trying to interfere with a running acquisition
215	Failed to switch devices into a safe state
216	Check spectrum did not succeed
217	Failed to set analyzer parameter
218	Unknown device command specified
219	Failed to load or operate direct device command
220	Unknown device specified

6 Version History

1.22 (2024-08-23, Issue #8274)

- New Commands: `GetAllDevices`, `GetAllLiveDeviceParameterNames`, `GetLiveDeviceParameterInfo`, `GetLiveDeviceParameterValue`

1.21 (2024-01-15, Issue #11566)

- `CreateDirectDeviceCommand` supports template groups

1.20 (2023-06-15, Issue #11159)

- Renamed document

1.19 (2022-07-11, Issue #10605)

- Support for “AbscissaRange” through `GetSpectrumDataInfo`
- Description of KREIOS MM k-space data from `GetAcquisitionData`

1.18 (2021-10-21, Issue #10101)

- New Commands: `CreateDirectDeviceCommand`, `GetDirectDeviceCommandInfo`, `GetDirectDeviceParameterInfo`, `GetDirectDeviceParameterValue`, `SetDirectDeviceParameterValue`, `ExecuteDirectDeviceCommand`

1.17 (2021-03-11, Issue #8583)

- Clarified and reformatted documentation

1.16 (2019-06-12, Issue #8273)

- New Command: `ValidateAnalyzerParameterValueDirectly`

1.15 (2019-04-02, Issue #8146)

- Clarified documentation of `SetAnalyzerParameterValueDirectly`
  (protocol version remains 1.14)

1.14 (2018-06-06, Issue #7166)

- New Commands: `GetAllDeviceCommands`, `GetAllDeviceParameterNames`, `GetDeviceParameterInfo`, `GetDeviceParameterValue`, `SetDeviceParameterValue`

1.13 (2017-07-18)

- Fixed Description of `GetSpectrumDataInfo`
  (protocol version remains 1.12)

1.12 (2017-01-24, Issue #6029):

- New Command: `SetAnalyzerParameterValueDirectly`

1.11 (2016-02-26, Issue #5428):

- New Command: `GetSpectrumDataInfo`
- Information about LabVIEW example added (2016-06-16, Issue #5671)

1.10 (2015-07-02, Issue #4830):

- New Commands: `CheckSpectrumFAT`, `CheckSpectrumSFAT`, `CheckSpectrumFRR`, `CheckSpectrumFE`

1.9 (2015-05-07, Issue #4384):

- New Command: `SetSafeState`

1.8 (2015-03-24, Issue #4063):

- New Command: `GetSpectrumParameterInfo`

1.7 (2014-12-17, Issue #3821):

- Added option: `SetSafeStateAfter` to `Start` command

1.6 (2014-12-17, Issue #3999):

- Bugfix regarding boolean parameters

1.5 (2014-11-13, Issue #4065):

- New Command: `GetAnalyzerVisibleName`

1.4 (2014-10-31, Issue #4149):

- Improved command `Connect`
- Security issues and bug fixes regarding concurrent measurements

1.3 (2014-10-14, Issue #4062):

- New Commands: `DefineSpectrumFRR`, `DefineSpectrumFE`
- Security issues and bug fixes regarding concurrent measurements

1.2 (2014-09-09, Issue #3820):

- New Command: `SetAnalyzerParameterVoltage`
- General documentation

1.1 (2014-09-09, Issue #3383):

- First fully documented version