

Programmers Reference

Bruyton DataAccess 9.0

Systat Software Electrophysiology Module 9.0

Bruyton

Contents: Overview

1 Introduction	12	7 PatchMaster: Visual Basic	200
1.1 Contents	12	7.1 Using DataAccess	200
1.2 Terms and Conditions	12	7.2 Getting Started	200
1.3 Versions	12	7.3 Robust Processing	203
1.4 Installation and Use	13	7.4 Operations	204
		7.5 Reference	206
2 ABF: IGOR Pro	14	8 Pulse: IGOR Pro	247
2.1 Script Interface	14	8.1 Script Interface	247
2.2 Robust Processing	16	8.2 Robust Processing	250
2.3 Operations	17	8.3 Operations	251
2.4 Reference	19	8.4 Reference	252
3 ABF: Visual Basic	54	9 Pulse: Visual Basic	286
3.1 Using DataAccess	54	9.1 Using DataAccess	286
3.2 Getting Started	54	9.2 Getting Started	286
3.3 Robust Processing	56	9.3 Robust Processing	289
3.4 Operations	57	9.4 Operations	290
3.5 Reference	59	9.5 Reference	292
4 Acquire: IGOR Pro	96	10 SON: Igor Pro	327
4.1 Script Interface	96	10.1 Script Interface	327
4.2 Robust Processing	100	10.2 Robust Processing	328
4.3 Operations	101	10.3 Operations	329
4.4 Reference	102	10.4 Reference	330
5 Acquire: Visual Basic	125	11 SON: Visual Basic	346
5.1 Using DataAccess	125	11.1 Using DataAccess	346
5.2 Getting Started	125	11.2 Getting Started	346
5.3 Robust Processing	129	11.3 Robust Processing	348
5.4 Operations	130	11.4 Operations	348
5.5 Reference	131	11.5 Reference	350
6 PatchMaster: IGOR Pro	156		
6.1 Script Interface	156		
6.2 Robust Processing	158		
6.3 Operations	159		
6.4 Reference	161		

Contents: Detail

1 Introduction	12
1.1 Contents	12
1.2 Terms and Conditions	12
1.2.1 Sales and Support	12
1.2.2 Copyright	12
1.2.3 US Government Restricted Rights	12
1.2.4 Trademarks	12
1.3 Versions	12
1.3.1 DataAccess	12
1.3.2 Systat Software Electrophysiology Module	12
1.4 Installation and Use	13
1.4.1 Microsoft Visual Basic	13
1.4.2 Systat Software SigmaPlot	13
1.4.3 WaveMetrics IGOR Pro	13
2 ABF: IGOR Pro	14
2.1 Script Interface	14
2.1.1 Opening a File	14
2.1.2 File Parameters	14
2.1.3 Episodes	14
2.1.4 Channels	15
2.1.5 Reading Data	15
2.1.6 Closing a File	16
2.2 Robust Processing	16
2.2.1 Errors	16
2.2.2 Status	16
2.3 Operations	17
2.3.1 Common Operations	17
2.3.2 File Operations	17
2.3.3 File Parameter Operations	17
2.3.4 Episode Operations	17
2.3.5 Stimulus Operations	18
2.3.6 Holding Potential Operations	18
2.3.7 Channel Operations	18
2.3.8 Epoch Operations	18
2.3.9 Leak Operations	18
2.3.10 Conditioning Pulse Train (CPT) Operations	18
2.3.11 Run Operations	19
2.3.12 Tag Operations	19
2.3.13 Annotation Operations	19
2.4 Reference	19
2.4.1 ABFAnnotationGetCount	20
2.4.2 ABFAnnotationGetText	20
2.4.3 ABFChannelGetADC	21
2.4.4 ABFChannelGetCount	21
2.4.5 ABFChannelGetFilterHigh	22
2.4.6 ABFChannelGetFilterLow	22
2.4.7 ABFChannelGetName	23
2.4.8 ABFChannelGetRange	23
2.4.9 ABFChannelGetUnits	24
2.4.10 ABFCPTGetBaseline	24
2.4.11 ABFCPTGetCount	25
2.4.12 ABFCPTGetPostTrain	25
2.4.13 ABFCPTGetStep	26
2.4.14 ABFEpisodeGetCount	26
2.4.15 ABFEpisodeGetDuration	27
2.4.16 ABFEpisodeGetSampleCount	27
2.4.17 ABFEpisodeGetStart	28
2.4.18 ABFEpisodeRead	28
2.4.19 ABFEpisodeReadStimulus	29
2.4.20 ABFEpisodeSet	30
2.4.21 ABFEpochGet	30
2.4.22 ABFEpochGetPrototype	31
2.4.23 ABFFileClose	32
2.4.24 ABFFileGetComment	33
2.4.25 ABFFileGetCreator	33
2.4.26 ABFFileGetDigital	34
2.4.27 ABFFileGetExperimentType	34
2.4.28 ABFFileGetIdentifier	35
2.4.29 ABFFileGetOperationMode	35
2.4.30 ABFFileGetSampleInterval	36
2.4.31 ABFFileGetSecondInterval	36
2.4.32 ABFFileGetTime	37
2.4.33 ABFFileGetTrigger	38
2.4.34 ABFFileGetVersion	39
2.4.35 ABFFileOpen	39
2.4.36 ABFFileParametersGet	40
2.4.37 ABFFileParametersRead	41
2.4.38 ABFGetStatusText	41
2.4.39 ABFGetVersion	42

Contents : Detail

2.4.40	ABFHoldingFinalGetAmplitude	42
2.4.41	ABFHoldingFinalGetDuration	43
2.4.42	ABFHoldingInitialGetAmplitude	43
2.4.43	ABFHoldingInitialGetDuration	44
2.4.44	ABFLeakGetCount	44
2.4.45	ABFLeakGetEnabled	45
2.4.46	ABFLeakGetHoldingLevel	45
2.4.47	ABFLeakGetPosition	46
2.4.48	ABFLeakGetPolarity	46
2.4.49	ABFLeakGetPulseInterval	47
2.4.50	ABFLeakGetSettlingTime	47
2.4.51	ABFRunGetAlgorithm	48
2.4.52	ABFRunGetCount	48
2.4.53	ABFRunGetInterval	49
2.4.54	ABFStimulusGetName	49
2.4.55	ABFStimulusGetRange	50
2.4.56	ABFStimulusGetSource	50
2.4.57	ABFStimulusGetUnits	51
2.4.58	ABFTagGetComment	51
2.4.59	ABFTagGetCount	52
2.4.60	ABFTagGetTime	52
2.4.61	ABFTagGetType	53
2.4.62	ABFTagGetVoiceNumber	53

3 ABF: Visual Basic 54

3.1 Using DataAccess 54

3.1.1	Status	54
3.1.2	Data Types	54
3.1.3	Calls	54

3.2 Getting Started 54

3.2.1	Opening a Session	54
3.2.2	Opening a File	55
3.2.3	File Parameters	55
3.2.4	Episodes	55
3.2.5	Channels	55
3.2.6	Reading Data	55
3.2.7	Closing a File	56

3.3 Robust Processing 56

3.4 Operations 57

3.4.1	Common Operations	57
3.4.2	Session Operations	57
3.4.3	File Operations	57
3.4.4	File Parameter Operations	57
3.4.5	Episode Operations	58
3.4.6	Stimulus Operations	58
3.4.7	Holding Potential Operations	58
3.4.8	Channel Operations	58
3.4.9	Epoch Operations	58
3.4.10	Leak Operations	59
3.4.11	Conditioning Pulse Train (CPT) Operations	59
3.4.12	Run Operations	59
3.4.13	Tag Operations	59
3.4.14	Annotation Operations	59

3.5 Reference 59

3.5.1	ABFAnnotationGetCount	60
3.5.2	ABFAnnotationGetText	60
3.5.3	ABFChannelGetADC	61
3.5.4	ABFChannelGetCount	61
3.5.5	ABFChannelGetFilterHigh	62
3.5.6	ABFChannelGetFilterLow	62
3.5.7	ABFChannelGetName	63
3.5.8	ABFChannelGetRange	63
3.5.9	ABFChannelGetSignalConditioner	64
3.5.10	ABFChannelGetUnits	64
3.5.11	ABFCPTGetBaseline	65
3.5.12	ABFCPTGetCount	65
3.5.13	ABFCPTGetPostTrain	66
3.5.14	ABFCPTGetStep	66
3.5.15	ABFEpisodeGetCount	67
3.5.16	ABFEpisodeGetDuration	67
3.5.17	ABFEpisodeGetSampleCount	68
3.5.18	ABFEpisodeGetStart	68
3.5.19	ABFEpisodeRead	69
3.5.20	ABFEpisodeReadStimulus	70
3.5.21	ABFEpisodeSet	70
3.5.22	ABFEpochGet	71
3.5.23	ABFEpochGetPrototype	72
3.5.24	ABFFileClose	73
3.5.25	ABFFileGetComment	73
3.5.26	ABFFileGetCreator	74
3.5.27	ABFFileGetDigital	74
3.5.28	ABFFileGetExperimentType	75
3.5.29	ABFFileGetIdentifier	75
3.5.30	ABFFileGetOperationMode	76
3.5.31	ABFFileGetSampleInterval	76
3.5.32	ABFFileGetSecondInterval	77
3.5.33	ABFFileGetSignature	78
3.5.34	ABFFileGetTime	78
3.5.35	ABFFileGetTrigger	79
3.5.36	ABFFileGetVersion	80
3.5.37	ABFFileOpen	80
3.5.38	ABFFileParametersGet	81
3.5.39	ABFFileParametersRead	82
3.5.40	ABFGetStatusText	82
3.5.41	ABFGetVersion	83
3.5.42	ABFHoldingFinalGetAmplitude	83
3.5.43	ABFHoldingFinalGetDuration	84
3.5.44	ABFHoldingInitialGetAmplitude	84
3.5.45	ABFHoldingInitialGetDuration	85
3.5.46	ABFLeakGetCount	85
3.5.47	ABFLeakGetEnabled	86
3.5.48	ABFLeakGetHoldingLevel	86
3.5.49	ABFLeakGetPosition	87
3.5.50	ABFLeakGetPolarity	87
3.5.51	ABFLeakGetPulseInterval	88
3.5.52	ABFLeakGetSettlingTime	88
3.5.53	ABFRunGetAlgorithm	89
3.5.54	ABFRunGetCount	89
3.5.55	ABFRunGetInterval	90
3.5.56	ABFSessionClose	90

Contents : Detail

- 3.5.57 ABFSessionOpen 91
- 3.5.58 ABFStimulusGetName 91
- 3.5.59 ABFStimulusGetRange 92
- 3.5.60 ABFStimulusGetSource 92
- 3.5.61 ABFStimulusGetUnits 93
- 3.5.62 ABFTagGetComment 93
- 3.5.63 ABFTagGetCount 94
- 3.5.64 ABFTagGetTime 94
- 3.5.65 ABFTagGetType 95
- 3.5.66 ABFTagGetVoiceNumber 95

4 Acquire: IGOR Pro 96

4.1 Script Interface 96

- 4.1.1 Opening a File 96
- 4.1.2 File Parameters 96
- 4.1.3 Series 96
- 4.1.4 Sweeps 97
- 4.1.5 Segments 97
- 4.1.6 Channels 98
- 4.1.7 Control Channels 98
- 4.1.8 Reading Data 98
- 4.1.9 Markers 99
- 4.1.10 Closing a File 99
- 4.1.11 Error Handling 99
- 4.1.12 Further Information 100

4.2 Robust Processing 100

- 4.2.1 Errors 100
- 4.2.2 Status 100

4.3 Operations 101

- 4.3.1 Common Operations 101
- 4.3.2 File Operations 101
- 4.3.3 File Parameter Operations 101
- 4.3.4 Channel Operations 101
- 4.3.5 Series Operations 101
- 4.3.6 Sweep Operations 101
- 4.3.7 Segment Operations 102
- 4.3.8 Block Operations 102
- 4.3.9 Control Channel Operations 102
- 4.3.10 Marker Operations 102

4.4 Reference 102

- 4.4.1 AcquireBlockGetCount 103
- 4.4.2 AcquireBlockGetSampleCount 103
- 4.4.3 AcquireChannelGetCount 104
- 4.4.4 AcquireChannelGetDescription 104
- 4.4.5 AcquireChannelGetRange 105
- 4.4.6 AcquireChannelGetScaling 105
- 4.4.7 AcquireControlGet 106
- 4.4.8 AcquireControlGetCount 107
- 4.4.9 AcquireControlGetHoldingValue 108
- 4.4.10 AcquireFileClose 108
- 4.4.11 AcquireFileGetComment 109
- 4.4.12 AcquireFileGetSampleRate 109
- 4.4.13 AcquireFileGetTechnique 110

- 4.4.14 AcquireFileGetTime 110
- 4.4.15 AcquireFileGetVersion 111
- 4.4.16 AcquireFileOpen 111
- 4.4.17 AcquireGetStatusText 112
- 4.4.18 AcquireGetVersion 112
- 4.4.19 AcquireMarkerGet 113
- 4.4.20 AcquireMarkerGetCount 114
- 4.4.21 AcquireMarkerGetData 114
- 4.4.22 AcquireMarkerGetExtendedData 115
- 4.4.23 AcquireMarkerGetText 115
- 4.4.24 AcquireSegmentGetCount 116
- 4.4.25 AcquireSegmentGetSampleCount 116
- 4.4.26 AcquireSegmentGetStart 117
- 4.4.27 AcquireSegmentRead 117
- 4.4.28 AcquireSegmentSet 118
- 4.4.29 AcquireSeriesGetComment 119
- 4.4.30 AcquireSeriesGetCount 119
- 4.4.31 AcquireSeriesGetStart 120
- 4.4.32 AcquireSeriesGetTechnique 120
- 4.4.33 AcquireSeriesSet 121
- 4.4.34 AcquireSweepGetComment 121
- 4.4.35 AcquireSweepGetCount 122
- 4.4.36 AcquireSweepGetDuration 122
- 4.4.37 AcquireSweepGetScreen 123
- 4.4.38 AcquireSweepGetStart 123
- 4.4.39 AcquireSweepGetTechnique 124
- 4.4.40 AcquireSweepSet 124

5 Acquire: Visual Basic 125

5.1 Using DataAccess 125

- 5.1.1 Status 125
- 5.1.2 Data Types 125
- 5.1.3 Calls 125

5.2 Getting Started 125

- 5.2.1 Opening a Session 125
- 5.2.2 Opening a File 126
- 5.2.3 File Parameters 126
- 5.2.4 Series 126
- 5.2.5 Sweeps 126
- 5.2.6 Segments 127
- 5.2.7 Channels 127
- 5.2.8 Control Channels 127
- 5.2.9 Reading Data 127
- 5.2.10 Markers 128
- 5.2.11 Closing a File 129
- 5.2.12 Error Handling 129
- 5.2.13 Further Information 129

5.3 Robust Processing 129

- 5.3.1 Status 129

5.4 Operations 130

- 5.4.1 Common Operations 130
- 5.4.2 Session Operations 130
- 5.4.3 File Operations 130

Contents : Detail

5.4.4	File Parameter Operations	130
5.4.5	Series Operations	130
5.4.6	Sweep Operations	130
5.4.7	Segment Operations	131
5.4.8	Channel Operations	131
5.4.9	Block Operations	131
5.4.10	Control Operations	131
5.4.11	Marker Operations	131

5.5 Reference 131

5.5.1	AcquireBlockGetCount	132
5.5.2	AcquireBlockGetSampleCount	132
5.5.3	AcquireBlockMaxGetSampleCount	133
5.5.4	AcquireChannelGetCount	133
5.5.5	AcquireChannelGetDescription	134
5.5.6	AcquireChannelGetRange	134
5.5.7	AcquireChannelGetScaling	135
5.5.8	AcquireControlGet	136
5.5.9	AcquireControlGetCount	137
5.5.10	AcquireControlGetHoldingValue	137
5.5.11	AcquireFileClose	138
5.5.12	AcquireFileGetComment	138
5.5.13	AcquireFileGetSampleRate	139
5.5.14	AcquireFileGetSignature	139
5.5.15	AcquireFileGetTechnique	140
5.5.16	AcquireFileGetTime	140
5.5.17	AcquireFileGetVersion	141
5.5.18	AcquireFileOpen	141
5.5.19	AcquireGetStatusText	142
5.5.20	AcquireGetVersion	142
5.5.21	AcquireMarkerGet	143
5.5.22	AcquireMarkerGetCount	144
5.5.23	AcquireMarkerGetData	144
5.5.24	AcquireMarkerGetExtendedData	145
5.5.25	AcquireMarkerGetText	145
5.5.26	AcquireSegmentGetCount	146
5.5.27	AcquireSegmentGetSampleCount	146
5.5.28	AcquireSegmentGetStart	147
5.5.29	AcquireSegmentRead	147
5.5.30	AcquireSegmentSet	148
5.5.31	AcquireSeriesGetComment	148
5.5.32	AcquireSeriesGetCount	149
5.5.33	AcquireSeriesGetStart	149
5.5.34	AcquireSeriesGetTechnique	150
5.5.35	AcquireSeriesSet	150
5.5.36	AcquireSessionClose	151
5.5.37	AcquireSessionOpen	151
5.5.38	AcquireSweepGetComment	152
5.5.39	AcquireSweepGetCount	152
5.5.40	AcquireSweepGetDuration	153
5.5.41	AcquireSweepGetScreen	153
5.5.42	AcquireSweepGetStart	154
5.5.43	AcquireSweepGetTechnique	154
5.5.44	AcquireSweepSet	155

6 PatchMaster: IGOR Pro 156

6.1 Script Interface 156

6.1.1	Opening a Data Set	156
6.1.2	File Parameters	156
6.1.3	Groups	156
6.1.4	Series	157
6.1.5	Sweeps	157
6.1.6	Segments	157
6.1.7	Channels	158
6.1.8	Reading Data	158
6.1.9	Closing a Data Set	158

6.2 Robust Processing 158

6.2.1	Errors	159
6.2.2	Status	159

6.3 Operations 159

6.3.1	Common Operations	159
6.3.2	File Operations	160
6.3.3	File Parameter Operations	160
6.3.4	Group Operations	160
6.3.5	Series Operations	160
6.3.6	Sweep Operations	160
6.3.7	Channel Operations	161
6.3.8	Stimulus Operations	161
6.3.9	Segment Operations	161

6.4 Reference 161

6.4.1	PMChannelGetADC	162
6.4.2	PMChannelGetAmplifier	162
6.4.3	PMChannelGetAvailable	163
6.4.4	PMChannelGetBandwidth	163
6.4.5	PMChannelGetCellPotential	164
6.4.6	PMChannelGetCompression	164
6.4.7	PMChannelGetCount	165
6.4.8	PMChannelGetLabel	166
6.4.9	PMChannelGetLeakAvailable	166
6.4.10	PMChannelGetLinkedStimulus	167
6.4.11	PMChannelGetMarked	167
6.4.12	PMChannelGetPipetteResistance	168
6.4.13	PMChannelGetRange	168
6.4.14	PMChannelGetRecordingMode	169
6.4.15	PMChannelGetSampleInterval	169
6.4.16	PMChannelGetSealResistance	170
6.4.17	PMChannelGetStart	170
6.4.18	PMChannelGetUnits	171
6.4.19	PMFileClose	171
6.4.20	PMFileGetComment	172
6.4.21	PMFileGetMarked	172
6.4.22	PMFileGetTime	173
6.4.23	PMFileGetPMVersion	173
6.4.24	PMFileGetVersion	174
6.4.25	PMFileOpen	174
6.4.26	PMGetStatusText	175
6.4.27	PMGetVersion	175
6.4.28	PMGroupGetComment	176
6.4.29	PMGroupGetCount	176

Contents : Detail

6.4.30	PMGroupGetEperiment	177
6.4.31	PMGroupGetLabel	177
6.4.32	PMGroupGetMarked	178
6.4.33	PMGroupSet	178
6.4.34	PMSegmentGetCount	179
6.4.35	PMSegmentGetParameters	179
6.4.36	PMSegmentGetWaveform	180
6.4.37	PMSeriesGetComment	181
6.4.38	PMSeriesGetCount	181
6.4.39	PMSeriesGetLabel	182
6.4.40	PMSeriesGetLeak	182
6.4.41	PMSeriesGetMarked	183
6.4.42	PMSeriesGetSampleInterval	184
6.4.43	PMSeriesGetStart	184
6.4.44	PMSeriesGetTime	185
6.4.45	PMSeriesGetUserName	185
6.4.46	PMSeriesGetUserParameter	186
6.4.47	PMSeriesSet	186
6.4.48	PMStimulusGetCount	187
6.4.49	PMStimulusGetDAC	187
6.4.50	PMStimulusGetRange	188
6.4.51	PMStimulusGetRelevantSegments	188
6.4.52	PMStimulusGetSource	189
6.4.53	PMStimulusGetStartSegment	189
6.4.54	PMStimulusGetTriggerType	190
6.4.55	PMStimulusGetUnits	190
6.4.56	PMSweepGetCount	191
6.4.57	PMSweepGetDigitalIn	191
6.4.58	PMSweepGetHoldingPotential	192
6.4.59	PMSweepGetLabel	192
6.4.60	PMSweepGetMarked	193
6.4.61	PMSweepGetRecordedCount	193
6.4.62	PMSweepGetSampleCount	194
6.4.63	PMSweepGetSolutions	194
6.4.64	PMSweepGetStart	195
6.4.65	PMSweepGetTemperature	195
6.4.66	PMSweepGetTimer	196
6.4.67	PMSweepGetUserParameter	196
6.4.68	PMSweepRead	197
6.4.69	PMSweepReadLeak	197
6.4.70	PMSweepReadRaw	198
6.4.71	PMSweepReadStimulus	198
6.4.72	PMSweepSet	199
7	PatchMaster: Visual Basic	200
7.1	Using DataAccess	200
7.1.1	Status	200
7.1.2	Data Types	200
7.1.3	Values	200
7.1.4	Calls	200
7.2	Getting Started	200
7.2.1	Opening a Session	201
7.2.2	Opening a Data Set	201
7.2.3	File Parameters	201
7.2.4	Groups	201
7.2.5	Series	201
7.2.6	Sweeps	202
7.2.7	Segments	202
7.2.8	Channels	202
7.2.9	Reading Data	202
7.2.10	Closing a Data Set	203
7.2.11	Closing a Session	203
7.3	Robust Processing	203
7.4	Operations	204
7.4.1	Common Operations	204
7.4.2	Session Operations	204
7.4.3	File Operations	204
7.4.4	File Parameter Operations	204
7.4.5	Group Operations	204
7.4.6	Series Operations	205
7.4.7	Sweep Operations	205
7.4.8	Channel Operations	205
7.4.9	Stimulus Operations	205
7.4.10	Segment Operations	206
7.5	Reference	206
7.5.1	PatchMasterChannelGetADC	206
7.5.2	PatchMasterChannelGetAmplifier	207
7.5.3	PatchMasterChannelGetAvailable	207
7.5.4	PatchMasterChannelGetBandwidth	208
7.5.5	PatchMasterChannelGetCellPotential	208
7.5.6	PatchMasterChannelGetCompression	209
7.5.7	PatchMasterChannelGetCount	210
7.5.8	PatchMasterChannelGetLabel	210
7.5.9	PatchMasterChannelGetLeakAvailable	211
7.5.10	PatchMasterChannelGetLinkedStimulus	211
7.5.11	PatchMasterChannelGetMarked	212
7.5.12	PatchMasterChannelGetPipetteResistance	212
7.5.13	PatchMasterChannelGetRange	213
7.5.14	PatchMasterChannelGetRecordingMode	213
7.5.15	PatchMasterChannelGetSampleInterval	214
7.5.16	PatchMasterChannelGetSealResistance	214
7.5.17	PatchMasterChannelGetStart	215
7.5.18	PatchMasterChannelGetUnits	215
7.5.19	PatchMasterFileClose	216
7.5.20	PatchMasterFileGetComment	216
7.5.21	PatchMasterFileGetMarked	217
7.5.22	PatchMasterFileGetSignature	217
7.5.23	PatchMasterFileGetTime	218
7.5.24	PatchMasterFileGetPatchMasterVersion	218
7.5.25	PatchMasterFileGetVersion	219
7.5.26	PatchMasterFileOpen	219
7.5.27	PatchMasterGetStatusText	220
7.5.28	PatchMasterGetVersion	220
7.5.29	PatchMasterGroupGetComment	221
7.5.30	PatchMasterGroupGetCount	221
7.5.31	PatchMasterGroupGetEperiment	222
7.5.32	PatchMasterGroupGetLabel	222
7.5.33	PatchMasterGroupGetMarked	223
7.5.34	PatchMasterGroupSet	223

Contents : Detail

7.5.35	PatchMasterSegmentGetCount	224
7.5.36	PatchMasterSegmentGetParameters	224
7.5.37	PatchMasterSegmentGetWaveform	225
7.5.38	PatchMasterSeriesGetComment	226
7.5.39	PatchMasterSeriesGetCount	226
7.5.40	PatchMasterSeriesGetLabel	227
7.5.41	PatchMasterSeriesGetLeak	227
7.5.42	PatchMasterSeriesGetMarked	228
7.5.43	PatchMasterSeriesGetSampleInterval	229
7.5.44	PatchMasterSeriesGetStart	229
7.5.45	PatchMasterSeriesGetTime	230
7.5.46	PatchMasterSeriesGetUserName	230
7.5.47	PatchMasterSeriesGetUserParameter	231
7.5.48	PatchMasterSeriesSet	231
7.5.49	PatchMasterSessionClose	232
7.5.50	PatchMasterSessionOpen	232
7.5.51	PatchMasterStimulusGetCount	233
7.5.52	PatchMasterStimulusGetDAC	233
7.5.53	PatchMasterStimulusGetRange	234
7.5.54	PatchMasterStimulusGetRelevantSegments	234
7.5.55	PatchMasterStimulusGetSource	235
7.5.56	PatchMasterStimulusGetStartSegment	235
7.5.57	PatchMasterStimulusGetTriggerType	236
7.5.58	PatchMasterStimulusGetUnits	236
7.5.59	PatchMasterSweepGetCount	237
7.5.60	PatchMasterSweepGetDigitalIn	237
7.5.61	PatchMasterSweepGetHoldingPotential	238
7.5.62	PatchMasterSweepGetLabel	238
7.5.63	PatchMasterSweepGetMarked	239
7.5.64	PatchMasterSweepGetRecordedCount	239
7.5.65	PatchMasterSweepGetSampleCount	240
7.5.66	PatchMasterSweepGetSolutions	240
7.5.67	PatchMasterSweepGetStart	241
7.5.68	PatchMasterSweepGetTemperature	241
7.5.69	PatchMasterSweepGetTimer	242
7.5.70	PatchMasterSweepGetUserParameter	242
7.5.71	PatchMasterSweepRead	243
7.5.72	PatchMasterSweepReadLeak	244
7.5.73	PatchMasterSweepReadRaw	245
7.5.74	PatchMasterSweepReadStimulus	245
7.5.75	PatchMasterSweepSet	246

8 Pulse: IGOR Pro 247

8.1 Script Interface 247

8.1.1	Opening a Data Set	247
8.1.2	File Parameters	247
8.1.3	Groups	247
8.1.4	Series	248
8.1.5	Sweeps	248
8.1.6	Segments	248
8.1.7	Channels	249
8.1.8	Reading Data	249
8.1.9	Closing a Data Set	250

8.2 Robust Processing 250

8.2.1	Errors	250
8.2.2	Status	250

8.3 Operations 251

8.3.1	Common Operations	251
8.3.2	File Operations	251
8.3.3	File Parameter Operations	251
8.3.4	Group Operations	251
8.3.5	Series Operations	251
8.3.6	Sweep Operations	252
8.3.7	Channel Operations	252
8.3.8	Stimulus Operations	252
8.3.9	Segment Operations	252

8.4 Reference 252

8.4.1	PulseChannelGetADC	253
8.4.2	PulseChannelGetCount	253
8.4.3	PulseChannelGetRange	254
8.4.4	PulseChannelGetUnits	254
8.4.5	PulseFileClose	255
8.4.6	PulseFileGetComment	255
8.4.7	PulseFileGetPulseVersion	256
8.4.8	PulseFileGetTime	256
8.4.9	PulseFileGetVersion	257
8.4.10	PulseFileOpen	257
8.4.11	PulseGetStatusText	258
8.4.12	PulseGetVersion	258
8.4.13	PulseGroupGetComment	259
8.4.14	PulseGroupGetCount	259
8.4.15	PulseGroupGetExperiment	260
8.4.16	PulseGroupGetLabel	260
8.4.17	PulseGroupSet	261
8.4.18	PulseSegmentGetCount	261
8.4.19	PulseSegmentGetParameters	262
8.4.20	PulseSegmentGetRelevant	263
8.4.21	PulseSegmentGetTrigger	263
8.4.22	PulseSegmentGetWaveform	264
8.4.23	PulseSeriesGetBackgroundNoise	264
8.4.24	PulseSeriesGetBandwidth	265
8.4.25	PulseSeriesGetCellPotential	265
8.4.26	PulseSeriesGetComment	266
8.4.27	PulseSeriesGetCount	266
8.4.28	PulseSeriesGetHoldingPotential	267
8.4.29	PulseSeriesGetLabel	267
8.4.30	PulseSeriesGetLeak	268
8.4.31	PulseSeriesGetPipette	269
8.4.32	PulseSeriesGetPotentialType	269
8.4.33	PulseSeriesGetRecordingMode	270
8.4.34	PulseSeriesGetSampleInterval	270
8.4.35	PulseSeriesGetSealResistance	271
8.4.36	PulseSeriesGetSolutions	271
8.4.37	PulseSeriesGetStart	272
8.4.38	PulseSeriesGetTemperature	272
8.4.39	PulseSeriesGetTime	273
8.4.40	PulseSeriesGetUserParameter1	273
8.4.41	PulseSeriesGetUserParameter2	274

Contents : Detail

8.4.42	PulseSeriesSet	274
8.4.43	PulseStimulusGetDAC	275
8.4.44	PulseStimulusGetRange	275
8.4.45	PulseStimulusGetTrigger	276
8.4.46	PulseStimulusGetTriggerCount	277
8.4.47	PulseStimulusGetUnits	277
8.4.48	PulseSweepGetAmplifier	278
8.4.49	PulseSweepGetCount	278
8.4.50	PulseSweepGetHoldingPotential	279
8.4.51	PulseSweepGetLabel	279
8.4.52	PulseSweepGetLeakAvailable	280
8.4.53	PulseSweepGetOnlineResult	280
8.4.54	PulseSweepGetSampleCount	281
8.4.55	PulseSweepGetStart	281
8.4.56	PulseSweepGetTimer	282
8.4.57	PulseSweepGetUserParameter1	282
8.4.58	PulseSweepGetUserParameter2	283
8.4.59	PulseSweepRead	283
8.4.60	PulseSweepReadLeak	284
8.4.61	PulseSweepReadRaw	284
8.4.62	PulseSweepReadStimulus	285
8.4.63	PulseSweepSet	285

9 Pulse: Visual Basic 286

9.1 Using DataAccess 286

9.1.1	Status	286
9.1.2	Data Types	286
9.1.3	Values	286
9.1.4	Calls	286

9.2 Getting Started 286

9.2.1	Opening a Session	287
9.2.2	Opening a Data Set	287
9.2.3	File Parameters	287
9.2.4	Groups	287
9.2.5	Series	287
9.2.6	Sweeps	288
9.2.7	Segments	288
9.2.8	Channels	288
9.2.9	Reading Data	288
9.2.10	Closing a Data Set	289
9.2.11	Closing a Session	289

9.3 Robust Processing 289

9.4 Operations 290

9.4.1	Common Operations	290
9.4.2	Session Operations	290
9.4.3	File Operations	290
9.4.4	File Parameter Operations	290
9.4.5	Group Operations	290
9.4.6	Series Operations	291
9.4.7	Sweep Operations	291
9.4.8	Channel Operations	291
9.4.9	Stimulus Operations	291
9.4.10	Segment Operations	292

9.5 Reference 292

9.5.1	PulseChannelGetADC	292
9.5.2	PulseChannelGetCount	293
9.5.3	PulseChannelGetRange	293
9.5.4	PulseChannelGetUnits	294
9.5.5	PulseFileClose	294
9.5.6	PulseFileGetComment	295
9.5.7	PulseFileGetPulseVersion	295
9.5.8	PulseFileGetSignature	296
9.5.9	PulseFileGetTime	296
9.5.10	PulseFileGetVersion	297
9.5.11	PulseFileOpen	297
9.5.12	PulseGetStatusText	298
9.5.13	PulseGetVersion	298
9.5.14	PulseGroupGetComment	299
9.5.15	PulseGroupGetCount	299
9.5.16	PulseGroupGetExperiment	300
9.5.17	PulseGroupGetLabel	300
9.5.18	PulseGroupSet	301
9.5.19	PulseSegmentGetCount	301
9.5.20	PulseSegmentGetParameters	302
9.5.21	PulseSegmentGetRelevant	303
9.5.22	PulseSegmentGetTrigger	303
9.5.23	PulseSegmentGetWaveform	304
9.5.24	PulseSeriesGetBackgroundNoise	304
9.5.25	PulseSeriesGetBandwidth	305
9.5.26	PulseSeriesGetCellPotential	305
9.5.27	PulseSeriesGetComment	306
9.5.28	PulseSeriesGetCount	306
9.5.29	PulseSeriesGetHoldingPotential	307
9.5.30	PulseSeriesGetLabel	307
9.5.31	PulseSeriesGetLeak	308
9.5.32	PulseSeriesGetPipette	309
9.5.33	PulseSeriesGetPotentialType	309
9.5.34	PulseSeriesGetRecordingMode	310
9.5.35	PulseSeriesGetSampleInterval	310
9.5.36	PulseSeriesGetSealResistance	311
9.5.37	PulseSeriesGetSolutions	311
9.5.38	PulseSeriesGetStart	312
9.5.39	PulseSeriesGetTemperature	312
9.5.40	PulseSeriesGetTime	313
9.5.41	PulseSeriesGetUserParameter1	313
9.5.42	PulseSeriesGetUserParameter2	314
9.5.43	PulseSeriesSet	314
9.5.44	PulseSessionClose	315
9.5.45	PulseSessionOpen	315
9.5.46	PulseStimulusGetDAC	316
9.5.47	PulseStimulusGetRange	316
9.5.48	PulseStimulusGetTrigger	317
9.5.49	PulseStimulusGetTriggerCount	318
9.5.50	PulseStimulusGetUnits	318
9.5.51	PulseSweepGetAmplifier	319
9.5.52	PulseSweepGetCount	319
9.5.53	PulseSweepGetHoldingPotential	320
9.5.54	PulseSweepGetLabel	320
9.5.55	PulseSweepGetLeakAvailable	321
9.5.56	PulseSweepGetOnlineResult	321

Contents : Detail

9.5.57	PulseSweepGetSampleCount	322
9.5.58	PulseSweepGetStart	322
9.5.59	PulseSweepGetTimer	323
9.5.60	PulseSweepGetUserParameter1	323
9.5.61	PulseSweepGetUserParameter2	324
9.5.62	PulseSweepRead	324
9.5.63	PulseSweepReadLeak	325
9.5.64	PulseSweepReadRaw	325
9.5.65	PulseSweepReadStimulus	326
9.5.66	PulseSweepSet	326
10	SON: Igor Pro	327
10.1	Script Interface	327
10.1.1	Opening a File	327
10.1.2	File Parameters	327
10.1.3	Segments	327
10.1.4	Channels	328
10.1.5	Reading Data	328
10.1.6	Closing a File	328
10.2	Robust Processing	328
10.2.1	Errors	328
10.2.2	Status	329
10.3	Operations	329
10.3.1	Common Operations	329
10.3.2	File Operations	329
10.3.3	File Parameter Operations	330
10.3.4	Segment Operations	330
10.3.5	Channel Operations	330
10.3.6	Marker Operations	330
10.4	Reference	330
10.4.1	SONChannelGetComment	331
10.4.2	SONChannelGetCount	331
10.4.3	SONChannelGetLabel	332
10.4.4	SONChannelGetMaximumTime	332
10.4.5	SONChannelGetPreTrigger	333
10.4.6	SONChannelGetSampleInterval	333
10.4.7	SONChannelGetType	334
10.4.8	SONChannelGetUnits	334
10.4.9	SONFileClose	335
10.4.10	SONFileGetComment	335
10.4.11	SONFileGetTime	336
10.4.12	SONFileGetVersion	336
10.4.13	SONFileOpen	337
10.4.14	SONGetStatusText	337
10.4.15	SONGetVersion	338
10.4.16	SONMarkerGetCount	338
10.4.17	SONMarkerGetCurve	339
10.4.18	SONMarkerGetCurveCount	340
10.4.19	SONMarkerGetCurveSize	340
10.4.20	SONMarkerGetData	341
10.4.21	SONMarkerGetNext	341
10.4.22	SONMarkerGetText	342
10.4.23	SONMarkerGetTimes	342
10.4.24	SONMarkerGetValueCount	343
10.4.25	SONMarkerGetValues	343
10.4.26	SONSegmentGetCount	344
10.4.27	SONSegmentGetRange	344
10.4.28	SONSegmentRead	345
11	SON: Visual Basic	346
11.1	Using DataAccess	346
11.1.1	Status	346
11.1.2	Data Types	346
11.1.3	Calls	346
11.2	Getting Started	346
11.2.1	Opening a Session	346
11.2.2	Opening a File	347
11.2.3	File Parameters	347
11.2.4	Segments	347
11.2.5	Channels	347
11.2.6	Reading Data	347
11.2.7	Closing a File	348
11.3	Robust Processing	348
11.4	Operations	348
11.4.1	Common Operations	349
11.4.2	Session Operations	349
11.4.3	File Operations	349
11.4.4	File Parameter Operations	349
11.4.5	Segment Operations	349
11.4.6	Channel Operations	349
11.4.7	Marker Operations	349
11.5	Reference	350
11.5.1	SONChannelGetComment	350
11.5.2	SONChannelGetCount	351
11.5.3	SONChannelGetLabel	351
11.5.4	SONChannelGetMaximumTime	352
11.5.5	SONChannelGetPreTrigger	352
11.5.6	SONChannelGetSampleInterval	353
11.5.7	SONChannelGetType	353
11.5.8	SONChannelGetUnits	354
11.5.9	SONFileClose	354
11.5.10	SONFileGetComment	355
11.5.11	SONFileGetSignature	355
11.5.12	SONFileGetTime	356
11.5.13	SONFileGetVersion	356
11.5.14	SONFileOpen	357
11.5.15	SONMarkerGetCount	357
11.5.16	SONMarkerGetCurve	358
11.5.17	SONMarkerGetCurveCount	359
11.5.18	SONMarkerGetCurveSize	359
11.5.19	SONMarkerGetData	360
11.5.20	SONMarkerGetNext	360
11.5.21	SONMarkerGetText	361
11.5.22	SONMarkerGetTimes	361
11.5.23	SONMarkerGetValueCount	362

Contents : Detail

11.5.24	SONMarkerGetValues	362
11.5.25	SONGetStatusText	363
11.5.26	SONGetVersion	363
11.5.27	SONSessionClose	364
11.5.28	SONSessionOpen	364
11.5.29	SONSegmentGetCount	365
11.5.30	SONSegmentGetRange	365
11.5.31	SONSegmentRead	366

1 Introduction

<hr/>	
Section	
Contents	p. 12
Terms and Conditions	p. 12
Versions	p. 12
Installation and Use	p. 13

1.1 Contents

This manual describes how to use the programming interface supplied by DataAccess.

1.2 Terms and Conditions

1.2.1 Sales and Support

DataAccess is produced by Bruxton Corporation, which can be reached at:

BRUXTON CORPORATION
7723 24th Ave NW
Seattle, WA 98117
USA

Voice: (206) 782-8862
FAX: (206) 782-8207

Web: <http://www.bruxton.com>

email: sales@bruxton.com
support@bruxton.com

1.2.2 Copyright

This manual is copyright 1995-2005 by Bruxton Corporation. All rights reserved.

The DataAccess program is copyright 1995-2005 by Bruxton Corporation.

1.2.3 US Government Restricted Rights

Any provision of DataAccess to the U.S. Government is with “Restricted Rights” as follows: Use, duplication, or disclosure by the Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause at FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Any provision of DataAccess documentation to the U.S. Government is with Limited Rights. Contractor/manufacturer is Bruxton Corporation, PO Box 17904, Seattle WA 98107, USA.

1.2.4 Trademarks

‘Bruxton’ is a registered trademark of Bruxton Corporation.

1.3 Versions

DataAccess is shipped as two different versions.

1.3.1 DataAccess

DataAccess includes support for all scripting interfaces.

1.3.2 Systat Software Electrophysiology Module

Systat Software Electrophysiology Module includes sup-

port for Visual Basic scripting only and all file formats .

1.4 Installation and Use

1.4.1 Microsoft Visual Basic

For each import filter, the associated DLL file must be installed in a directory that is on the path when running Visual Basic. The DLL file associated with each import filter is shown in table 1.

Table 1 Visual Basic import filter DLL files

Filter	File
Axon ABF	ABFVB.dll
Bruyton Acquire	AcquireVB.dll
CED SON	SONVB.dll
HEKA PatchMaster	PatchMasterVB.dll
HEKA Pulse	PulseVB.dll

To make DataAccess available to your program, add the associated bas file to your project. The bas file associated with each import filter is shown in table 2.

Table 2 Visual Basic import filter bas files

Filter	File
Axon ABF	ABFVB.bas
Bruyton Acquire	AcquireVB.bas
CED SON	SONVB.bas
HEKA PatchMaster	PatchMasterVB.bas
HEKA Pulse	PulseVB.bas

1.4.2 Systat Software SigmaPlot

Scripting for Systat Software SigmaPlot uses the Microsoft Visual Basic scripting support in DataAccess. For each import filter, the associated DLL file must be installed in a directory that is on the path when running SigmaPlot. The DLL file associated with each import filter is shown in table 3.

Table 3 SigmaPlot import filter DLL files

Filter	File
Axon ABF	ABFVB.dll
Bruyton Acquire	AcquireVB.dll

Table 3 SigmaPlot import filter DLL files

Filter	File
CED SON	SONVB.dll
HEKA PatchMaster	PatchMasterVB.dll
HEKA Pulse	PulseVB.dll

To make DataAccess available in a module, include the associated bas file in the module. Use the modules located in the "SigmaPlotVB" subdirectory. The SigmaPlot version of Visual Basic does not support passing of arrays by type. Instead they must be declared "As Any". The required statement is shown in table 4.

Table 4 SigmaPlot statements to use scripting

Filter	Declaration
Axon ABF	'#Uses "ABFVB.bas"
Bruyton Acquire	'#Uses "AcquireVB.bas"
CED SON	'#Uses "SONVB.bas"
HEKA PatchMaster	'#Uses "PatchMasterVB.bas"
HEKA Pulse	'#Uses "PulseVB.bas"

The leading single-quote (!) at the beginning of the line is required. The .bas file must be in the same folder as your SigmaPlot project, or you must explicitly specify the path to the .bas file.

1.4.3 WaveMetrics IGOR Pro

For each import filter, the associated xop file must be present in the "IGOR Extensions" folder. The xop files used are shown in table 5.

Table 5 IGOR Pro xop files

Filter	XOP
Axon ABF	ABFXOP.xop
Bruyton Acquire	AcquireXOP.xop
CED SON	SONXOP.xop
HEKA PatchMaster	PatchMasterXOP.xop
HEKA Pulse	PulseXOP.xop

2 ABF: IGOR Pro

Section	
2.1 Script Interface	P. 14
2.2 Robust Processing	P. 16
2.3 Operations	P. 17
2.4 Reference	P. 19

2.1 Script Interface

This section explains how to access an ABF data file using the commands provided by DataAccess. It contains a step by step description of the operations to perform to open a file and read the data in it.

2.1.1 Opening a File

Open an ABF data file using the ABFFileOpen operation.
☞ ABFFileOpen, p. 39.

In order to open a data file, you must supply the full path to the file. The IGOR PRO “Open” command can be used to bring up a file selection dialog and supply the resulting file path.

The following example brings up a file selection dialog to obtain a file specification. It then uses ABFFileOpen to open the file for processing:

```
Variable fileReferenceNumber
Variable status

Open /D/R/T="BINFTEXT" fileReferenceNumber

if (strlen(S_fileName) > 0)
  ABFFileOpen S_fileName, status
endif
```

2.1.2 File Parameters

Once you have opened a file, you can determine the parameters of the file. The parameters include the creation date and time, the file comment, and the sampling rate.

☞ File Parameter Operations, p. 17.

The following example obtains a set of file parameters:

```
Variable status
String fileDateTime
ABFFileGetTime fileDateTime, status
Print fileDateTime

String fileComment
ABFFileGetComment fileComment, status
Print fileComment

Variable sampleInterval
ABFFileGetSampleInterval sampleInterval, status
Print sampleInterval
```

2.1.3 Episodes

A file is a sequence of episodes. You use the episode operations to select an episode and determine its parameters.
☞ Episode Operations, p. 17.

You use ABFEpisodeGetCount to determine the number of episodes in a file. *☞ ABFEpisodeGetCount, p. 26.*

Episodes are numbered 1 to N, where N is the number of episodes in a file. You use ABFEpisodeSet to set the episode to use. *☞ ABFEpisodeSet, p. 30.*

The following example determines the duration of each

episode in a file. The duration is measured in seconds:

```
Variable status
Variable episode_count
Variable current_episode
Variable duration

ABFepisodeGetCount episode_count

if (episode_count > 0)
  current_episode = 1
  do
    ABFepisodeSet current_episode, status
    ABFepisodeGetDuration duration, status
    current_episode += 1
  while (current_episode <= episode_count)
else
  | The file contains no episodes
endif
```

A file may have no episodes if no data was acquired.

2.1.4 Channels

ABFChannelGetCount returns the number of channels recorded in a file. [↗ ABFChannelGetCount, p. 21.](#)

The following example obtains the number of channels recorded in a file:

```
Variable status
Variable channel_count

ABFChannelGetCount channel_count, status
```

Channel numbers range from 0 to N-1, where N is the number of channels in the file.

2.1.5 Reading Data

To read an episode, use ABFepisodeRead. ABFepisodeRead can read either a whole episode or sections of an episode. [↗ ABFepisodeRead, p. 28.](#)

Normally one would read whole episodes but it may be necessary to read sections of an episode if you have insufficient memory available to read whole episodes.

The wave into which you read data must contain double-precision values.

Reading Episodes

The following example reads each episode into a wave.

```
Variable status
Variable episode_count
Variable current_episode
Variable episode_size

ABFepisodeGetCount episode_count, status

if (episode_count > 0)
  current_episode = 1
  do
    ABFepisodeSet current_episode, status
    ABFepisodeGetSampleCount episode_size, status
    Make/D/N=(episode_size) data

    ABFepisodeRead channel, 0, episode_size, data,
      status
    | Data processing goes here
    current_episode += 1
  while (current_episode <= episode_count)
else
  | The file has no episodes
endif
```

Episodes and Channels

Each channel in an episode has the same number of data samples.

The following example reads data from each channel of the current episode into a wave.

```
Variable status
Variable channel_count
Variable channel
Variable episode_size

ABFepisodeGetSize episode_size, status

Make/D/N=(episode_size) data

ABFChannelGetCount channel_count, status

channel = 0
do
  ABFepisodeRead channel, 0, episode_size, data,
```

```

    status
    | Data processing goes here
    channel += 1
    while (channel < channel_count)

```

Reading Sections of an Episode

The following example reads the first “length” values of the first episode into a wave.

```

Variable status
Variable episode_size

ABFEpisodeGetSampleCount episode_size, status

if (episode_size < length)
    | Limit the read to the actual length of the episode
    length = episode_size
endif

Make/D/N=(length) data

ABFEpisodeRead channel, 0, length, data, status

```

2.1.6 Closing a File

To close a file, use `ABFFileClose`. [☞ ABFFileClose, p. 32.](#)

The following example closes an ABF file.

```
ABFFileClose
```

2.2 Robust Processing

if you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

This section explains how to use the information provided by `DataAccess` to recover from error conditions in your Igor Pro procedures.

2.2.1 Errors

The status information provided by `DataAccess` allows you to recover from error conditions caused by the program, user or the system. It does not allow you to recover from syntax errors in your program.

Program Errors

Errors in your program include such mistakes as using `ABFEpisodeSet` to set an invalid episode number. You can determine the range of valid episode numbers using `ABFEpisodeGetCount`, so you can avoid setting an invalid episode number. Catching and reporting such errors is helpful in debugging a procedure.

System Errors

System errors include such problems as an error reading an ABF data file. Such errors are usually the result of problems such as a corrupted disk, a missing floppy or CD-ROM, or defective hardware such as a disk drive. Catching such errors prevents further problems and may allow the procedure to recover, for example by allowing the user to insert a missing file medium.

User Errors

User errors include mistakes such as when the user selects a file that is not an ABF data file. The recovery allows your procedures to provide an error indication to the user and allow the user to select a different file.

Syntax Errors

Syntax errors include mistakes such as passing the wrong number or types of parameters to an operation. An improperly dimensioned wave is also treated as a syntax error. Such errors are returned directly to Igor Pro and cause a procedure to terminate.

2.2.2 Status

Nearly all operations return a status value through a status parameter. The status value is non-zero if an error occurred. All `DataAccess` and system errors are returned in this manner. Such errors will not terminate a procedure. The procedure should be designed to handle such conditions. Igor Pro errors, such as a missing parameter, are returned directly to Igor and will terminate a procedure.

Status values for specific errors are not defined, and may change from version to version of `DataAccess`. The `ABFGetStatusText` operation translates a status value to text. [☞ ABFGetStatusText, p. 41.](#)

The following example allows the user to select a file, and

does not terminate until the user selects a valid ABF file.

```

Variable fileNumber
Variable status
String message

do
  Open /D/R/T="BINFTEXT" fileNumber

  if (strlen(S_fileName) > 0)
    ABFFileOpen S_fileName, status

    if (status != 0)
      ABFGetStatusText status, message
      Print "ABFFileOpen error: ", message
    endif
  endif
while (status != 0)

```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

2.3 Operations

This chapter describes each of the operations provided by DataAccess. The operations are grouped by category.

2.3.1 Common Operations

The common operations are valid regardless of whether a file is open or not. The common operations are shown in table 6.

Table 6 Common operations

Operation	
ABFGetVersion	p. 42
ABFGetStatusText	p. 41

2.3.2 File Operations

The file operations allow you to access an ABF data file.

The file operations are shown in table 7.

Table 7 File operations

Operation	
ABFFileOpen	p. 39
ABFFileClose	p. 32

2.3.3 File Parameter Operations

The file parameter operations allow you to obtain file parameters. The file parameter operations are shown in table 8. The operations are valid when a file is open.

Table 8 File parameter operations

Operation	
ABFFileGetComment	p. 33
ABFFileGetCreator	p. 33
ABFFileGetDigital	p. 34
ABFFileGetExperimentType	p. 34
ABFFileGetOperationMode	p. 35
ABFFileGetIdentifier	p. 35
ABFFileGetSampleInterval	p. 36
ABFFileGetSecondInterval	p. 36
ABFFileGetTime	p. 37
ABFFileGetTrigger	p. 38
ABFFileGetVersion	p. 39
ABFFileParametersGet	p. 40
ABFFileParametersRead	p. 41

2.3.4 Episode Operations

The episode operations allow you to access each episode of a file. The episode operations are shown in table 9. The operations are valid when a file is open. Most require that a current episode be set.

Table 9 Episode operations

Operation	
ABFEpisodeGetCount	p. 26
ABFEpisodeSet	p. 30
ABFEpisodeGetDuration	p. 27
ABFEpisodeGetSampleCount	p. 27
ABFEpisodeGetStart	p. 28
ABFEpisodeRead	p. 28
ABFEpisodeReadStimulus	p. 29

2.3.5 Stimulus Operations

The stimulus operations allow you to determine the output data source and the data used for output. The stimulus operations are shown in table 10. The operations are valid when a file is open. Most require that a current episode be set.

Table 10 Stimulus operations

Operation	
ABFStimulusGetName	p. 49
ABFStimulusGetRange	p. 50
ABFStimulusGetSource	p. 50
ABFStimulusGetUnits	p. 51

If the epoch parameters are used to generate a stimulus, you can use the epoch operations to determine the epoch parameters. *Epoch Operations*, p. 18.

2.3.6 Holding Potential Operations

The holding potential operations allow you to determine the amplitude and duration of the holding potential. The holding potential operations are shown in table 11. The operations are valid when a file is open.

Table 11 Holding potential operations

Operation	
ABFHoldingInitialGetAmplitude	p. 43
ABFHoldingInitialGetDuration	p. 44
ABFHoldingFinalGetAmplitude	p. 42
ABFHoldingFinalGetDuration	p. 43

2.3.7 Channel Operations

The channel operations allow you to determine the number of channels in a file, and obtain channel parameters. The channel operations are shown in table 12. The operations are valid when a file is open.

Table 12 Channel operations

Operation	
ABFChannelGetCount	p. 21
ABFChannelGetADC	p. 21
ABFChannelGetFilterHigh	p. 22
ABFChannelGetFilterLow	p. 22

Table 12 Channel operations

Operation	
ABFChannelGetName	p. 23
ABFChannelGetRange	p. 23
ABFChannelGetUnits	p. 24

2.3.8 Epoch Operations

The epoch operations allow you to determine the parameters used for stimulation. The epoch operations are shown in table 13.

Table 13 Epoch operations

Operation	
ABFEpochGet	p. 30
ABFEpochGetPrototype	p. 31

The epoch operations are valid when a file is open. ABFEpochGet requires that a current episode be set.

2.3.9 Leak Operations

The leak operations allow you to determine the leak subtraction if used.

The leak operations are shown in table 14.

Table 14 Leak operations

Operation	
ABFLeakGetCount	p. 44
ABFLeakGetEnabled	p. 45
ABFLeakGetHoldingLevel	p. 45
ABFLeakGetPosition	p. 46
ABFLeakGetPolarity	p. 46
ABFLeakGetPulseInterval	p. 47
ABFLeakGetSettlingTime	p. 47

2.3.10 Conditioning Pulse Train (CPT) Operations

The conditioning pulse train (CPT) operations allow you to determine the parameters of the conditioning pulse train, if any, used during acquisition.

The conditioning pulse train operations are shown in

table 15.

Table 15 Conditioning Pulse Train operations

Operation	
ABFCPTGetCount	p. 25
ABFCPTGetBaseline	p. 24
ABFCPTGetPostTrain	p. 25
ABFCPTGetStep	p. 26

The conditioning pulse train operations are valid when a file is open.

2.3.11 Run Operations

The run operations allow you to determine the run averaging used to record an episode.

The run operations are shown in table 16.

Table 16 Run operations

Operation	
ABFRunGetAlgorithm	p. 48
ABFRunGetCount	p. 48
ABFRunGetInterval	p. 49

2.3.12 Tag Operations

The tag operations allow you to read the tags stored in a data file. The tag operations are shown in table 17. The operations are valid when a file is open.

Table 17 Tag operations

Operation	
ABFTagGetComment	p. 51
ABFTagGetCount	p. 52
ABFTagGetTime	p. 52
ABFTagGetType	p. 53
ABFTagGetVoiceNumber	p. 53

2.3.13 Annotation Operations

The annotation operations allow you to read the annotations stored in a data file. The annotation operations are shown in table 18. The operations are valid when a file is

open.

Table 18 Annotation operations

Operation	
ABFAnnotationGetCount	p. 20
ABFAnnotationGetText	p. 20

2.4 Reference

This section lists each DataAccess operation in alphabetical order.

2.4.1 ABFAnnotationGetCount

ABFAnnotationGetCount returns the number of annotations.

Syntax

ABFAnnotationGetCount *count, status*

count A variable to receive the number of annotations.

status A variable to receive the status value.

Example

Variable status
Variable count

ABFAnnotationGetCount count, status

Discussion

You must have a file open.

2.4.2 ABFAnnotationGetText

ABFAnnotationGetText returns the text associated with an annotation.

Syntax

ABFAnnotationGetText *annotation, text, status*

annotation A value representing the annotation number to return.

text A String variable in which the text is returned.

status A variable to receive the status value.

Example

Variable status
String text

ABFAnnotationGetText 0, text, status

Discussion

You must have a file open.

Annotations are numbered 0 to N-1, where N is the value returned by ABFAnnotationGetCount. [↗](#) *ABFAnnotationGetCount*, p. 20.

2.4.3 ABFChannelGetADC

ABFChannelGetADC returns the device channel number of the specified channel.

Syntax

ABFChannelGetADC *channel, adc, status*

channel A value representing the channel to use.

adc A variable to receive the channel number.

status A variable to receive the status value.

Example

Variable adc
Variable status

ABFChannelGetADC 1, adc, status

Discussion

You must have a file open.

Channel numbers are logical, and may not correspond to the channel number of a device. The adc value returned is the channel number on the front panel of the data acquisition device.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [☞ ABFChannelGetCount, p. 21.](#)

2.4.4 ABFChannelGetCount

ABFChannelGetCount returns the number of channels recorded in the current ABF data file.

Syntax

ABFChannelGetCount *count, status*

count A variable in which to store the number of channels.

status A variable to receive the status value.

Example

Variable status
Variable channel_count

ABFChannelGetCount channel_count, status

Discussion

You must have a file open.

The channel count is the number of active channels recorded in the ABF data file. For example, if data was recorded from channels 0, 2, and 5, the channel count is 3. The channel count must be at least 1.

2.4.5 ABFChannelGetFilterHigh

ABFChannelGetFilterHigh returns information about the high pass filter used.

Syntax

ABFChannelGetFilterHigh *channel, type, cutoff, status*

channel A value representing the channel to use.

type A variable to receive the high pass filter type.

cutoff A variable to receive the filter cutoff frequency.

status A variable to receive the status value.

Example

Variable status
Variable type
Variable cutoff

ABFChannelGetFilterHigh type, cutoff, status

Discussion

You must have a file open.

The cutoff frequency is measured in Hz. If the cutoff frequency is 0, the filter is bypassed. If the cutoff frequency is -1, the inputs are grounded.

The filter type is one of the values listed in table 19.

Table 19 Filter types

Value	Type
0	None
1	External
2	Simple
3	Bessel
4	Butterworth

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount.

☞ *ABFChannelGetCount, p. 21.*

2.4.6 ABFChannelGetFilterLow

ABFChannelGetFilterLow returns information about the low pass filter used.

Syntax

ABFChannelGetFilterLow *channel, type, cutoff, status*

channel A value representing the channel to use.

type A variable to receive the low pass filter type.

cutoff A variable to receive the filter cutoff frequency.

status A variable to receive the status value.

Example

Variable status
Variable type
Variable cutoff

ABFChannelGetFilterLow type, cutoff, status

Discussion

You must have a file open.

The cutoff frequency is measured in Hz. If the frequency is 100000, it means that filtering is bypassed.

The filter type is one of the values listed in table 20.

Table 20 Filter types

Value	Type
0	None
1	External
2	Simple
3	Bessel
4	Butterworth

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount.

☞ *ABFChannelGetCount, p. 21.*

2.4.7 ABFChannelGetName

ABFChannelGetName returns the name of the specified channel.

Syntax

ABFChannelGetName *channel, name, status*

channel A value representing the channel to use.

name A String variable in which the channel name is returned.

status A variable to receive the status value.

Example

Variable status
String text

ABFChannelGetName 0, text, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [☞ ABFChannelGetCount, p. 21.](#)

2.4.8 ABFChannelGetRange

ABFChannelGetRange returns the range of the specified channel.

Syntax

ABFChannelGetRange *channel, minimum, maximum, status*

channel The channel to use.

minimum The name of the variable to receive the lower limit.

maximum The name of the variable to receive the upper limit.

status A variable to receive the status value.

Example

Variable status
Variable minimum
Variable maximum

ABFChannelGetRange 0, minimum, maximum, status

Discussion

You must have a file open.

ABFChannelGetRange returns the minimum and maximum possible values for the specified channel.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [☞ ABFChannelGetCount, p. 21.](#)

2.4.9 ABFChannelGetUnits

ABFChannelGetUnits returns the units of the specified channel.

Syntax

ABFChannelGetUnits *channel, units, status*

channel A value representing the channel to use.

units A String variable in which the channel units are returned.

status A variable to receive the status value.

Example

Variable status

String text

ABFChannelGetUnits 0, units, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [↗ ABFChannelGetCount, p. 21.](#)

2.4.10 ABFCPTGetBaseline

ABFCPTGetBaseline returns conditioning pulse train pulse baseline information.

Syntax

ABFCPTGetBaseline *channel, duration, amplitude, status*

channel A value representing the stimulus channel to use.

duration A variable to receive the baseline duration.

amplitude A variable to receive the baseline amplitude.

status A variable to receive the status value.

Example

Variable status

Variable duration

Variable amplitude

ABFCPTGetBaseline 0, duration, amplitude, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

A conditioning pulse train pulse consists of a baseline segment and a step segment. ABFCPTGetStep returns information about the step segment. [↗ ABFCPTGetStep, p. 26.](#)

2.4.11 ABFCPTGetCount

ABFCPTGetCount returns the number of conditioning pulse train pulses.

Syntax

ABFCPTGetCount *channel, count, status*

channel A value representing the stimulus channel to use.

count A variable to receive the number of pulses.

status A variable to receive the status value.

Example

Variable status
Variable pulse_count

ABFCPTGetCount 0, pulse_count, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

2.4.12 ABFCPTGetPostTrain

ABFCPTGetPostTrain returns post conditioning pulse train information.

Syntax

ABFCPTGetPostTrain *channel, duration, amplitude, status*

channel A value representing the stimulus channel to use.

duration A variable to receive the post train duration.

amplitude A variable to receive the post train amplitude.

status A variable to receive the status value.

Example

Variable status
Variable duration
Variable amplitude

ABFCPTGetPostTrain 0, duration, amplitude, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

2.4.13 ABFCPTGetStep

ABFCPTGetStep returns conditioning pulse train pulse step information.

Syntax

ABFCPTGetStep *channel, duration, amplitude, status*

channel A value representing the stimulus channel to use.

duration A variable to receive the step duration.

amplitude A variable to receive the step amplitude.

status A variable to receive the status value.

Example

Variable status
Variable duration
Variable amplitude

ABFCPTGetStep duration, amplitude, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

A conditioning pulse train pulse consists of a baseline segment and a step segment. ABFCPTGetBaseline returns information about the baseline segment.
☞ *ABFCPTGetBaseline*, p. 24.

2.4.14 ABFEpisodeGetCount

ABFEpisodeGetCount returns the number of episodes recorded in the file.

Syntax

ABFEpisodeGetCount *count, status*

count A variable to receive the number of episodes.

status A variable to receive the status value.

Example

Variable status
Variable episode_count

ABFEpisodeGetCount episode_count, status

Discussion

You must have a file open.

The episode count is the number of episodes recorded in the ABF data file. Episodes are numbered 1 to N, where N is the number of episodes in the file.

2.4.15 ABFEpisodeGetDuration

ABFEpisodeGetDuration returns the duration of the current episode.

Syntax

ABFEpisodeGetDuration *duration, status*

duration A variable to receive the episode duration.

status A variable to receive the status value.

Example

Variable status
Variable duration

```
ABFEpisodeSet episode, status
ABFEpisodeGetDuration duration, status
```

Discussion

You must have a file open and an episode selected.
☞ *ABFEpisodeSet*, p. 30.

The duration of an episode is measured in seconds.

You must have a file open and an episode selected.
☞ *ABFEpisodeSet*, p. 30.

2.4.16 ABFEpisodeGetSampleCount

ABFEpisodeGetSampleCount returns the number of samples in the current episode.

Syntax

ABFEpisodeGetSampleCount *samples, status*

samples A variable to receive the number of samples in the episode.

status A variable to receive the status value.

Example

Variable status
Variable samples

```
ABFEpisodeGetSampleCount samples, status
```

Discussion

You must have a file open and an episode selected.
☞ *ABFEpisodeSet*, p. 30.

The number of samples in an episode is measured per channel. All channels have the same number of samples.

2.4.17 ABFEpisodeGetStart

ABFEpisodeGetStart returns the starting time of the current episode.

Syntax

ABFEpisodeGetStart *start, status*

start A variable to receive the episode starting time.

status A variable to receive the status value.

Example

Variable status
Variable start

ABFEpisodeSet episode, status
ABFEpisodeGetStart start, status

Discussion

You must have a file open and an episode selected.
☞ *ABFEpisodeSet*, p. 30.

The start time of an episode is measured in seconds since the beginning of the experiment.

2.4.18 ABFEpisodeRead

ABFEpisodeRead reads the data for one channel of a section of the current episode into a wave.

Syntax

ABFEpisodeRead *channel, start, length, data, status*

channel A value representing the channel to use.

start A value representing the first sample to read.

length A value representing the number of samples to read.

data A wave to receive the data. The wave must be double-precision and be long enough to contain the specified number of samples.

status A variable to receive the status value.

Example

Variable status
Variable samples
Variable start

ABFEpisodeGetSampleCount samples, status

samples = samples/2
start = samples/4

Make/D/N=(samples) data

ABFEpisodeRead channel, start, samples, data, status

Discussion

You must have a file open and an episode selected.
☞ *ABFEpisodeSet*, p. 30.

The start and length values must satisfy the following criteria:

- 1 The start value must be positive.
- 2 The length value must be greater than zero.
- 3 The sum of the start and length values must be less than the episode length.

To ensure that the parameters are in range, you can use `ABFEpisodeGetSampleCount` to determine the number of data values in the episode. [☞ ABFEpisodeGetSampleCount, p. 27.](#)

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use `ABFChannelGetCount`. [☞ ABFChannelGetCount, p. 21.](#)

2.4.19 ABFEpisodeReadStimulus

`ABFEpisodeReadStimulus` reads the output data for the current episode into a wave.

Syntax

`ABFEpisodeReadStimulus channel, data, status`

channel A value representing the stimulus channel to use.

data A wave to receive the data. The wave must be double-precision, and long enough to contain all the data in the episode.

status A variable to receive the status value.

Example

Variable status

Variable samples

`ABFEpisodeGetSampleCount samples, status`

`Make/D/N=(samples) data`

`ABFEpisodeReadStimulus 0, data, status`

Discussion

You must have a file open and an episode selected. [☞ ABFEpisodeSet, p. 30.](#)

The stimulus channels are numbered 0 to 1.

To ensure that the wave is long enough, you can use `ABFEpisodeGetSampleCount` to determine the number of data values in the episode. [☞ ABFEpisodeGetSampleCount, p. 27.](#)

The output data may be generated from the epoch information or provided as a sequence of samples (DAC file). In either case `ABFEpisodeReadStimulus` will provide the output data. To determine the source of the data, use `ABFStimulusGetSource`. [☞ ABFStimulusGetSource, p. 50.](#)

The returned data are in the user units. [☞ ABFStimulusGetUnits, p. 51.](#)

2.4.20 ABFEpisodeSet

ABFEpisodeSet selects the current episode.

Syntax

ABFEpisodeSet *episode, status*

episode A value representing the episode.

status A variable to receive the status value.

Example

Variable status
Variable episodes
Variable duration

ABFEpisodeGetCount episodes, status

ABFEpisodeGetDuration episodes, duration, status

Discussion

You must have a file open.

Episodes are numbered 1 to N, where N is the number of episodes in the file. ➦ *ABFEpisodeGetCount*, p. 26.

2.4.21 ABFEpochGet

ABFEpochGet returns the parameters for the specified epoch for the current episode.

Syntax

ABFEpochGet *channel, epoch, type, duration, amplitude, digital, status*

channel A value representing the stimulus channel to use.

epoch A value representing the epoch.

type A variable receive the type of the epoch.

duration A variable to receive the duration of the epoch in samples.

amplitude A variable to receive the amplitude of the epoch.

digital A variable to receive the digital outputs for the epoch.

status A variable to receive the status value.

Example

Variable type
Variable duration
Variable amplitude
Variable digital
Variable status

ABFEpochGet 0, 0, type, duration, amplitude, digital, status

Discussion

You must have a file open and an episode selected. ➦ *ABFEpisodeSet*, p. 30.

The stimulus channels are numbered 0 to 1.

Type values are shown in table 21.

Table 21 Epoch types

Value	Type
0	Disabled
1	Step
2	Ramp

The type of an epoch is fixed for all episodes.

The amplitude, duration and digital output used for an epoch may vary with each episode. ABFEpochGet returns the values for the current episode.

The duration is the number of samples output during the epoch.

The amplitude is in the units returned by ABFStimulusGetUnits. [↗ ABFStimulusGetUnits, p. 51.](#)

The digital value represents the digital output on/off states for the epoch.

Epochs are numbered 0 to 9.

2.4.22 ABFEpochGetPrototype

ABFEpochGetPrototype returns the information from which the epoch data is constructed.

Syntax

ABFEpochGetPrototype *channel, epoch, type, amplitude, amplitude_increment, duration, duration_increment, pulse_period, pulse_width, status*

channel A value representing the stimulus channel to use.

epoch A value representing the epoch.

type A variable to receive the epoch type.

amplitude A variable to receive the base amplitude of the epoch.

amplitude_increment A variable to receive the amplitude increment of the epoch.

duration A variable to receive the base duration of the epoch.

duration_increment A variable to receive the duration increment of the epoch.

pulse_period A variable to receive the pulse period for a pulse train epoch.

pulse_width A variable to receive the pulse width for a pulse train epoch.

status A variable to receive the status value.

Example

```
Variable status
Variable amplitude
Variable amplitude_increment
Variable duration
Variable duration_increment
Variable actual_amplitude
Variable actual_duration
Variable pulse_period
Variable pulse_width
```

```
ABFEpochGetPrototype 0, 0, amplitude,
amplitude_increment, duration, duration_increment,
```

pulse_period, pulse_width, status

actual_amplitude =
amplitude + (episode - 1) * amplitude_increment

actual_duration =
duration + (episode - 1) * duration_increment

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The duration of an epoch is measured in seconds.

When epochs are used to generate a stimulus, the actual epoch amplitude and duration are calculated using the equations shown in the example. That is, during the first episode, the base amplitude and base duration are used. During the second episode, the base amplitude plus the amplitude increment, and the base duration plus the duration increment are used.

The stimulus amplitude and duration might not match the parameters returned by `ABFEpochGetPrototype` in several cases:

- 1 The user list can be used to specify a set of values for an epoch amplitude or duration. The user list overrides the epoch prototype.
- 2 The file might use a DAC file instead of epochs to generate output. This can be determined using `ABFDACGetSource`. [☞ *ABFStimulusGetSource*, p. 50.](#)

The epoch prototype is independent of the episode.

The amplitudes are in the user units. [☞ *ABFStimulusGetUnits*, p. 51.](#)

Epochs are numbered 0 to 9.

Type values are shown in table 21.

2.4.23 ABFFileClose

`ABFFileClose` terminates processing of an ABF file.

Syntax

`ABFFileClose`

Example

`ABFFileClose`

Discussion

You must have a file open.

You should perform `ABFFileClose` as soon as you complete processing of a file.

2.4.24 ABFFileGetComment

ABFFileGetComment returns the comment for the file.

Syntax

ABFFileGetComment *comment, status*

comment A String variable in which the file comment is returned.

status A variable to receive the status value.

Example

Variable status
String text

ABFFileGetComment 0, text, status

Discussion

You must have a file open.

2.4.25 ABFFileGetCreator

ABFFileGetCreator returns the creator for the file.

Syntax

ABFFileGetCreator *creator, status*

creator A String variable in which the file creator is returned.

status A variable to receive the status value.

Example

Variable status
String text

ABFFileGetCreator 0, text, status

Discussion

You must have a file open.

2.4.26 ABFFileGetDigital

ABFFileGetDigital returns the digital output settings for the file.

Syntax

ABFFileGetCreator *enabled, holding, status*

enabled A variable to receive the enabled state of the digital outputs.

holding A variable to receive the digital holding values.

status A variable to receive the status value.

Example

Variable enabled
Variable holding

ABFFileGetDigital enabled holding, status

Discussion

You must have a file open.

2.4.27 ABFFileGetExperimentType

ABFFileGetExperimentType returns the type of experiment recorded in the file.

Syntax

ABFFileGetExperimentType *type, status*

type A variable to receive the experiment type.

status A variable to receive the status value.

Example

Variable status
Variable experiment_type

ABFFileGetExperimentType experiment_type, status

Discussion

You must have a file open.

The experiment types are shown in table 22.

Table 22 Experiment types

Value	Type
0	Voltage clamp
1	Current clamp

2.4.28 ABFFileGetIdentifier

ABFFileGetIdentifier returns cell identification values.

Syntax

ABFFileGetIdentifier *identifier, value, status*

identifier An value containing the identifier number.

value A variable to receive the identifier value.

status A variable to receive the status value.

Example

Variable status;
Variable identifier;

ABFFileGetIdentifier 1, identifier, status

Discussion

You must have a file open.

Identifiers are numbered 1 to 3. The identifier values are entered by the user.

2.4.29 ABFFileGetOperationMode

ABFFileGetOperationMode returns the mode used to record data in the file.

Syntax

ABFFileGetOperationMode *mode, status*

mode A variable to receive the operation mode.

status A variable to receive the status value.

Example

Variable status
Variable operation_mode

ABFFileGetOperationMode operation_mode, status

Discussion

You must have a file open.

The operation mode values are shown in table 23.

Table 23 Operation mode values

Value	Mode
1	Variable-length events
2	Fixed-length events
3	Gap-free
4	High-speed oscilloscope
5	Clampex

2.4.30 ABFFileGetSampleInterval

ABFFileGetSampleInterval returns the sampling interval in seconds on each channel.

Syntax

ABFFileGetSampleInterval *interval, status*

interval A variable to receive the sampling interval.

status A variable to receive the status value.

Example

Variable status
Variable interval

ABFFileGetSampleInterval interval, status

Discussion

You must have a file open.

All channels are sampled at the same interval. The interval measures the time between samples on the same channel.

The sampling interval is the same throughout the file, unless a second sampling interval is specified. If a second sampling interval is specified, the sampling interval may change during each episode. [↗ ABFFileGetSecondInterval, p. 36.](#)

The sampling interval is reported in units of seconds. It always represents an integer number of microseconds. To convert the sampling interval to the exact number of microseconds, use code similar to the following:

Variable interval
Variable microseconds

ABFFileGetSampleInterval interval

microseconds = round(interval * 1000000.0)

Rounding to the nearest integer value compensates for potential roundoff error in the representation of the sampling interval.

2.4.31 ABFFileGetSecondInterval

ABFFileGetSecondInterval returns a description of the second sampling interval, including the sample interval and the sample number in each episode from which sampling is performed using the second sampling interval.

Syntax

ABFFileGetSecondIntervalStart *interval, sample, status*

interval A variable to receive the second sampling interval.

sample A variable to receive the sample number from which sampling is performed using the second sampling interval.

status A variable to receive the status value.

Example

Variable status
Variable interval
Variable sample

ABFFileGetSecondInterval interval, sample, status

```
if (interval <> 0.0)
  | The second sampling interval is used
endif
```

Discussion

You must have a file open.

All channels are sampled at the same interval. The interval measures the time between samples on the same channel.

If the second sampling interval is non-zero, the sample number specifies the sample number within each episode from which the second sampling interval is used. Samples preceding the sample number are sampled using the normal sampling interval. [↗ ABFFileGetSampleInterval, p. 36.](#)

The second sampling interval is reported in units of seconds. It always represents an integer number of microseconds. To convert the second sampling interval to the exact

number of microseconds, use code similar to the following:

```
Variable status  
Variable interval  
Variable microseconds  
Variable sample
```

```
ABFFileGetSecondInterval interval, sample
```

```
microseconds = round(interval * 1000000.0)
```

Rounding to the nearest integer value compensates for potential roundoff error in the representation of the sampling interval.

2.4.32 ABFFileGetTime

ABFFileGetTime returns the creation date and time of the file.

Syntax

```
ABFFileGetTime date_time, status
```

date_time A String variable to receive the file creation date and time.

status A variable to receive the status value.

Example

```
Variable status  
String date_time
```

```
ABFFileGetTime date_time, status
```

Discussion

You must have a file open.

2.4.33 ABFFileGetTrigger

ABFFileGetTrigger returns the trigger parameters.

Syntax

```
ABFFileGetTrigger action, interval, polarity,
                 source, threshold, status
```

action A variable to receive the trigger action.

interval A variable to receive the trigger episode interval.

polarity A variable to receive the trigger polarity.

source A variable to receive the trigger source.

threshold A variable to receive the trigger threshold.

status A variable to receive the status value.

Example

```
Variable status
Variable trigger_action
Variable trigger_interval
Variable trigger_polarity
Variable trigger_source
Variable trigger_threshold
```

```
ABFFileGetTrigger trigger_action,
                 trigger_interval, trigger_polarity, trigger_source,
                 trigger_threshold, status
```

Discussion

You must have a file open.

The trigger action is one of the values in table 24.

Table 24 Trigger actions

Number	Action
0	Start one sweep
1	Start one run
2	Start one trial

The trigger source is one of the values in table 25.

Table 25 Trigger sources

Number	Trigger source
0 or positive	channel number
-1	external
-2	keyboard
-3	start to start interval

The episode interval is the time between episode starts. The episode interval is used only when the trigger source is 'start to start interval'.

2.4.34 ABFFileGetVersion

ABFFileGetVersion returns the ABF format version of the file.

Syntax

ABFFileGetVersion *version, status*

version A String variable in which the file version is returned.

status A variable to receive the status value.

Example

Variable status
String version

ABFFileGetVersion version, status

Discussion

You must have a file open.

2.4.35 ABFFileOpen

ABFFileOpen prepares an ABF data file for processing.

Syntax

ABFFileOpen *path, status*

path A String value representing the file path.

status A variable to receive the status value.

Example

Variable status

ABFFileOpen "Data.dat", status

Discussion

You must not have a file open.

Once a file is open, you can access file parameters. You can process the episodes in the file. [Episode Operations](#), p. 17.
[File Parameter Operations](#), p. 17.

When you are done with a file, you should close it.
[ABFFileClose](#), p. 32.

ABFFileOpen invalidates the current episode. You must set the current episode using ABFEpisodeSet.
[ABFEpisodeSet](#), p. 30.

2.4.36 ABFFileParametersGet

ABFFileParametersGet returns information about the user parameter list.

Syntax

ABFFileParametersGet *index, enabled, parameter, count, repeat, status*

index An int value indicating the parameter index.

enabled A variable to receive the enabled state of the user parameter list.

parameter A variable to receive the value indicating which parameter is varied.

count A variable to receive number of values contained in the list.

repeat A variable to receive the repeat mode.

status A variable to receive the status value.

Example

Variable enabled
Variable parameter
Variable count
Variable repeat

ABFFileParametersGet 0, enabled, parameter, count,
repeat, status

Discussion

You must have a file open.

Index can range from 0 to 3.

A parameter that varies from episode to episode may be overridden by values stored in the parameter list.

If the value returned for enabled is zero, no parameter value is returned. If the value returned for enabled is non-zero, a parameter value is returned. The possible parameter

values are listed in table 26.

Table 26 Parameters

Value	Varying parameter
0	number of conditioning pules
1	conditioning baseline duration
2	conditioning baseline level
3	conditioning step duration
4	conditioning step level
5	conditioning post train duration
6	conditioning post train level
7	episode start to start
8	inactive holding level
9	digital inter episode
10	number of P/N pulses
11-20	epoch digital value, epoch number value-11
21-30	epoch level, epoch number value-21
31-40	epoch duration, epoch number value-31
41-50	pulse train period, epoch number value-41
51-60	pulse train width, epoch number value-51

If count is less than the number of episodes in the file, and repeat is zero the last value in the list is used for the remainder of the episodes. Otherwise the list is repeated.

The parameter list is stored in the file as text, and can possibly contain invalid entries. In this case the count value returned is the number of valid entries. If the count value is zero, no entries are valid, and the parameter list is ignored.

Use ABFFileParametersRead to read the values in the parameter list. [↗ ABFFileParametersRead, p. 41.](#)

2.4.37 ABFFileParametersRead

ABFFileParametersRead returns the values in the user parameter list.

Syntax

ABFFileParametersRead *index, values, count, status*

index An int value indicating the parameter index.

values A wave to receive the parameter list values.

count A value indicating the number of parameter list values to read.

status A variable to receive the status value.

Example

Variable count

Variable status

ABFEpisodeGetCount count, status

Make/D/N=(count) parameters

ABFFileParametersRead 0, parameters, count, status

Discussion

You must have a file open.

Index can range from 0 to 3.

Use ABFFileParametersGet to get the number of values stored in the list. [↗ ABFFileParametersGet, p. 40.](#)

If count is larger than the actual number of values in the parameter list, the remaining variables in the array are filled with the last value in the parameter list. [↗ ABFEpisodeGetCount, p. 26.](#)

2.4.38 ABFGetStatusText

ABFGetStatusText translates a status value returned by a DataAccess operation to a text string.

Syntax

ABFGetStatusText *status, message*

status A value representing the status value to translate.

message A String variable in which the translated text is returned.

Example

Variable status

String message

ABFFileOpen "Sample.dat", status

if (status <> 0)

 ABFGetStatusText status, message

 Print "ABFFileOpen error: ", message

endif

Discussion

You need not have a file open.

ABFGetStatusText is the only means provided to interpret status values. [↗ Status, p. 16.](#)

2.4.39 ABFGetVersion

ABFGetVersion returns the version number of the DataAccess package.

Syntax

ABFGetVersion *version*

version A variable to receive the DataAccess version number.

Example

Variable version

ABFGetVersion version

Discussion

You need not have a file open.

If you are writing a set of procedures using DataAccess, you can use ABFGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess. You can use the following code to ensure that this version or a later version is in use:

```
Variable version
ABFGetVersion version
```

```
if (version < N)
  | Handle the error
endif
```

2.4.40 ABFHoldingFinalGetAmplitude

ABFHoldingFinalGetAmplitude returns the final holding amplitude of each episode.

Syntax

ABFHoldingFinalGetAmplitude *channel, amplitude, status*

channel A value representing the stimulus channel to use.

amplitude A variable to receive the holding amplitude.

status A variable to receive the status value.

Example

Variable status
Variable amplitude

ABFHoldingFinalGetAmplitude 0, amplitude, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The holding amplitude is the same for all episodes.

The amplitude is in user units. [↗ ABFStimulusGetUnits, p. 51.](#)

2.4.41 ABF Holding Final Get Duration

ABF Holding Final Get Duration returns the final holding duration of each episode.

Syntax

ABF Holding Final Get Duration *channel, duration, status*

channel A value representing the stimulus channel to use.

duration A variable to receive the holding duration.

status A variable to receive the status value.

Example

Variable status
Variable duration

ABF Holding Final Get Duration 0, duration, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The duration is measured in seconds.

The holding duration is the same for all episodes.

2.4.42 ABF Holding Initial Get Amplitude

ABF Holding Initial Get Amplitude returns the initial holding amplitude of each episode.

Syntax

ABF Holding Initial Get Amplitude *channel, amplitude, status*

channel A value representing the stimulus channel to use.

amplitude A variable to receive the holding amplitude.

status A variable to receive the status value.

Example

Variable status
Variable amplitude

ABF Holding Initial Get Amplitude 0, amplitude, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The holding amplitude is the same for all episodes.

The amplitude is in user units. [↗ ABFStimulusGetUnits, p. 51.](#)

2.4.43 ABF Holding Initial Get Duration

ABF Holding Initial Get Duration returns the initial holding duration of each episode.

Syntax

ABF Holding Initial Get Duration *channel, duration, status*

channel A value representing the stimulus channel to use.

duration A variable to receive the holding duration.

status A variable to receive the status value.

Example

Variable status
Variable duration

ABF Holding Initial Get Duration 0, duration, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The duration is measured in seconds.

The holding duration is the same for all episodes.

2.4.44 ABF Leak Get Count

ABF Leak Get Count returns the number of P/N subpulses.

Syntax

ABF Leak Get Count *channel, count, status*

channel A value representing the stimulus channel to use.

count A variable in which to store the number of subpulses.

status A variable to receive the status value.

Example

Variable status
Variable leak_count

ABF Leak Get Count 0, leak_count, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

2.4.45 ABFLeakGetEnabled

ABFLeakGetEnabled returns the P/N subtraction usage.

Syntax

ABFLeakGetEnabled *channel, enabled, status*

channel A value representing the stimulus channel to use.

enabled A variable in which to store the P/N subtraction usage.

status A variable to receive the status value.

Example

Variable status
Variable leak_enabled

ABFLeakGetEnabled0, leak_enabled, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

ABFLeakGetEnabled returns a non-zero value if P/N subtraction is enabled.

2.4.46 ABFLeakGetHoldingLevel

ABFLeakGetHoldingLevel returns the P/N holding level.

Syntax

ABFLeakGetHoldingLevel *channel, level, status*

channel A value representing the stimulus channel to use.

level A variable in which to store the P/N holding level.

status A variable to receive the status value.

Example

Variable status
Variable holding_level

ABFLeakGetHoldingLevel 0, holding_level, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The holding level is the output level between subpulses.

2.4.47 ABFLeakGetPosition

ABFLeakGetPosition returns the P/N position.

Syntax

ABFLeakGetPosition *position, status*

position A variable in which to store the P/N position.

status A variable to receive the status value.

Example

Variable status

Variable leak_position

ABFLeakGetPosition leak_position, status

Discussion

You must have a file open.

The possible values of the leak position are shown in table 27.

Table 27 Leak position values

Value	Position
0	Leak subpulses occur before main stimulus
1	Leak subpulses occur after main stimulus

2.4.48 ABFLeakGetPolarity

ABFLeakGetPolarity returns the P/N polarity.

Syntax

ABFLeakGetPolarity *channel, polarity, status*

channel A value representing the stimulus channel to use.

polarity A variable in which to store the P/N polarity.

status A variable to receive the status value.

Example

Variable status

Variable leak_polarity

ABFLeakGetPolarity 0, leak_polarity, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The possible values of the leak polarity are shown in table 28.

Table 28 Leak polarity values

Value	Polarity
-1	Leak subpulses have polarity opposite main stimulus
1	Leak subpulses have same polarity as main stimulus

2.4.49 ABFLeakGetPulseInterval

ABFLeakGetPulseInterval returns the P/N pulse interval.

Syntax

ABFLeakGetPulseInterval *interval, status*

interval A variable in which to store the P/N pulse interval.

status A variable to receive the status value.

Example

Variable status
Variable leak_interval

ABFLeakGetPulseInterval leak_interval, status

Discussion

You must have a file open.

The interval is measured in seconds. It is the time between subpulse starts.

2.4.50 ABFLeakGetSettlingTime

ABFLeakGetSettlingTime returns the P/N pulse settling time.

Syntax

ABFLeakGetSettlingTime *time, status*

time A variable in which to store the P/N settling time.

status A variable to receive the status value.

Example

Variable status
Variable settling_time

ABFLeakGetSettlingTime settling_time, status

Discussion

You must have a file open.

The settling time is measured in seconds. It is the time between the subpulses and the main stimulus.

2.4.51 ABFRunGetAlgorithm

ABFRunGetAlgorithm returns the algorithm used to average runs.

Syntax

ABFRunGetAlgorithm *algorithm, weighting, status*

algorithm A variable to receive the algorithm.

weighting A variable to receive the weighting factor.

status A variable to receive the status value.

Example

Variable status
Variable algorithm
Variable weighting

ABFRunGetAlgorithm algorithm, weighting, status

Discussion

You must have a file open.

The possible values of the algorithm are shown in table 29.

Table 29 Run algorithm values

Value	Algorithm
0	Cumulative averaging
1	Most recent (weighted) averaging

2.4.52 ABFRunGetCount

ABFRunGetCount returns the number of runs used to record data in the file.

Syntax

ABFRunGetCount *count, status*

count A variable to receive the run count.

status A variable to receive the status value.

Example

Variable status
Variable run_count

ABFRunGetCount run_count, status

Discussion

You must have a file open.

2.4.53 ABFRunGetInterval

ABFRunGetInterval returns the interval between runs.

Syntax

ABFRunGetInterval *interval, status*

interval A variable to receive the run interval.

status A variable to receive the status value.

Example

Variable status
Variable run_interval

```
ABFRunGetInterval run_interval, status
```

Discussion

You must have a file open.

The interval is measured in seconds. It is the time between run starts.

2.4.54 ABFStimulusGetName

ABFStimulusGetName returns the name associated with the stimulus.

Syntax

ABFStimulusGetName *channel, name, status*

channel A value representing the stimulus channel to use.

name A String variable to receive the stimulus name.

status A variable to receive the status value.

Example

Variable status
String name

```
ABFStimulusGetName 0, name, status
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

2.4.55 ABFStimulusGetRange

ABFStimulusGetRange returns the range of the stimulus.

Syntax

ABFStimulusGetRange *minimum, maximum, status*

minimum A variable to receive the lower limit.

maximum A variable to receive the upper limit.

status A variable to receive the status value.

Example

Variable status
Variable minimum
Variable maximum

ABFStimulusGetRange minimum,maximum, status

Discussion

You must have a file open.

ABFStimulusGetRange returns the minimum and maximum possible values for the stimulus.

2.4.56 ABFStimulusGetSource

ABFStimulusGetSource returns the source used to construct the stimulus.

Syntax

ABFStimulusGetSource *channel, source, status*

channel A value representing the stimulus channel to use.

source A variable to receive the source indicator.

status A variable to receive the status value.

Example

Variable status
Variable stimulus_source

ABFStimulusGetSource 0, stimulus_source, status

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The source indicator values are shown in table 30.

Table 30 Stimulus Source Indicator Values

Value	Meaning
0	No stimulus
1	Stimulus generated from epochs
2	Stimulus generated from DAC file

2.4.57 ABFStimulusGetUnits

ABFStimulusGetUnits returns the units for the stimulus.

Syntax

ABFStimulusGetUnits *channel, units, status*

channel A value representing the stimulus channel to use.

units A String variable to receive the units.

status A variable to receive the status value.

Example

Variable status
String stimulus_units

```
ABFStimulusGetUnits 0, stimulus_units, status
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The stimulus string is entered by the user during data acquisition.

2.4.58 ABFTagGetComment

ABFTagGetComment returns the comment associated with a tag.

Syntax

ABFTagGetComment *tag, comment, status*

tag A value representing the tag number to return.

comment A String variable in which the comment is returned.

status A variable to receive the status value.

Example

Variable status
String text

```
ABFTagGetComment 0, text, status
```

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. ↗ *ABFTagGetCount*, p. 52.

2.4.59 ABFTagGetCount

ABFTagGetCount returns the number of file tags.

Syntax

ABFTagGetCount *count, status*

count A variable to receive the number of tags.

status A variable to receive the status value.

Example

Variable status

Variable count

ABFTagGetCount count, status

Discussion

You must have a file open.

2.4.60 ABFTagGetTime

ABFTagGetTime returns the time associated with a tag.

Syntax

ABFTagGetTime *tag, time, status*

tag A value representing the tag number to return.

time A variable in which the tag time is returned.

status A variable to receive the status value.

Example

Variable status

Variable tag_time

ABFTagGetTime 0, tag_time, status

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. [↗ ABFTagGetCount, p. 52.](#)

The tag time is measured in seconds from the beginning of the experiment.

2.4.61 ABFTagGetType

ABFTagGetType returns the type of a tag.

Syntax

ABFTagGetType *tag, type, status*

tag A value representing the tag number to return.

type A variable in which the tag type is returned.

status A variable to receive the status value.

Example

Variable status

Variable tag_type

ABFTagGetType 0, tag_type, status

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. [↗ ABFTagGetCount, p. 52.](#)

The possible tag types are shown in table 31.

Table 31 Tag type values

Value	Type
0	Time
1	Comment
2	External
3	Voice

2.4.62 ABFTagGetVoiceNumber

ABFTagGetVoiceNumber returns the voice number of a tag.

Syntax

ABFTagGetVoiceNumber *tag, number, status*

tag A value representing the tag number to return.

number A variable in which the voice tag number is returned.

status A variable to receive the status value.

Example

Variable status

Variable voice_number

ABFTagGetVoiceNumber 0, voice_number, status

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. [↗ ABFTagGetCount, p. 52.](#)

The voice tag number is valid only if the associated tag is a voice tag. The tag type is returned by ABFTagGetType. [↗ ABFTagGetType, p. 53.](#)

3 ABF: Visual Basic

Section	
3.1 Using DataAccess	P. 54
3.2 Getting Started	P. 54
3.3 Robust Processing	P. 56
3.4 Operations	P. 57
3.5 Reference	P. 59

3.1 Using DataAccess

To make DataAccess available to a Visual Basic program, add the file ABFVB.bas to the project. Ensure that the file ABFVB.dll is in your path.

To make DataAccess available to an Systat Software SigmaPlot script module, include the following statement in the module:

```
'#Uses "ABFVB.bas"
```

The leading quote (!) is required. The bas file must be in the same folder as your SigmaPlot project, or you must explicitly specify the path to the bas file.

3.1.1 Status

Each operation returns a status code. This code is zero if no error is detected.

If an error is detected, the returned status value is non-zero. It can be translated to a text string using ABFGetStatusText.

3.1.2 Data Types

Integer values are passed as the data type “Long”. Real values are passed as the data type “Double”. Arrays are passed by passing the first element in the array. Character string values are passed as the data type “Variant”. The data type “String” is not used because when Visual Basic passes parameters to dlls, it performs undesired translations from Unicode to ASCII.

3.1.3 Calls

Almost all DataAccess procedures return a value. In Visual Basic, you have several choices of syntax when you call such procedures. For example, you can call ABFChannelGetCount as follows:

- 1 As a function, using the return value:

```
result = ABFChannelGetCount(session, count)
```

- 2 As a subroutine using “Call”:

```
Call ABFChannelGetCount(session, count)
```

- 3 Directly as a command. In this case, the parameters are not enclosed in parentheses:

```
ABFChannelGetCount session, count
```

Any of the methods work. The examples in this manual generally use the command form.

3.2 Getting Started

This section explains how to access an ABF data file. It contains a step by step description of the operations to perform to open a file and read the data in it.

3.2.1 Opening a Session

Open a session using the ABFSessionOpen operation.  *ABFSessionOpen, p. 91.*

ABFSessionOpen operation returns a handle. You must pass this handle to other operations that use that session.

3.2.2 Opening a File

Open an ABF data file using the ABFFileOpen operation. [↗ ABFFileOpen, p. 80.](#)

In order to open a data file, you must supply a file path.

The following example opens a session, then opens a file with the path “f:\test.dat”. After the file is open, it closes the file and the session.

```
Dim session As Long
Dim status As Long
Dim file_name As Variant
file_name = "f:\test.dat"

ABFSessionOpen session
status = ABFFileOpen(session, file_name)

If status <> 0 Then
    ' The file could not be opened. Handle the
    ' error and do not call ABFFileClose
End If

ABFFileClose session
ABFSessionClose session
```

3.2.3 File Parameters

Once you have opened a file, you can determine the parameters of the file. The parameters include the creation date and time, the file comment, and the sampling rate. [↗ File Parameter Operations, p. 57.](#)

The following example obtains a set of file parameters:

```
Dim file_date_time As Date
Dim interval As Double

ABFFileGetTime session, file_date_time
ABFFileGetSampleInterval session, interval
```

3.2.4 Episodes

A file is a sequence of episodes. You use the episode operations to select an episode and determine its parameters.

[↗ Episode Operations, p. 58.](#)

You use ABFEpisodeGetCount to determine the number of episodes in a file. [↗ ABFEpisodeGetCount, p. 67.](#)

Episodes are numbered 1 to N, where N is the number of episodes in a file. You use ABFEpisodeSet to set the episode to use. [↗ ABFEpisodeSet, p. 70.](#)

The following example determines the duration of each episode in a file. The duration is measured in seconds:

```
Dim episode_count As Long
Dim current_episode As Long
Dim duration As Double

ABFEpisodeGetCount session, episode_count

If episode_count > 0 Then
    For current_episode = 1 To episode_count
        ABFEpisodeSet session, current_episode
        ABFEpisodeGetDuration session, duration
        Next current_episode
    Else
        ' The file contains no episodes
    End If
```

A file may have no episodes if no data was acquired.

3.2.5 Channels

ABFChannelGetCount returns the number of channels recorded in a file. [↗ ABFChannelGetCount, p. 61.](#)

The following example obtains the number of channels recorded in a file:

```
Dim channel_count As Long

ABFChannelGetCount session, channel_count
```

Channel numbers range from 0 to N-1, where N is the number of channels in the file.

3.2.6 Reading Data

To read an episode, use ABFEpisodeRead. ABFEpisodeRead can read either a whole episode or sections of an episode. [↗ ABFEpisodeRead, p. 69.](#)

Normally one would read whole episodes but it may be necessary to read sections of an episode if you have insufficient memory available to read whole episodes.

Reading Episodes

The following example reads each episode into an array.

```
Dim buffer() As Double
Dim episode_count As Long
Dim current_episode As Long
Dim episode_size As Long

ABFEpisodeGetCount session, episode_count

If episode_count > 0 Then
  For current_episode = 1 To episode_count
    ABFEpisodeSet session, current_episode
    ABFEpisodeGetSampleCount
      session, episode_size
    ReDim buffer(episode_size-1)

    ABFEpisodeRead session, channel, 0,
      episode_size, buffer(0)
    ' Data processing goes here
  Next current_episode
Else
  ' The file has no episodes
End If
```

Episodes and Channels

Each channel in an episode has the same number of data samples.

The following example reads data from each channel of the current episode into an array.

```
Dim buffer() As Double
Dim channel_count As Long
Dim channel As Long
Dim episode_size As Long

ABFEpisodeGetSize session, episode_size

ReDim buffer(episode_size-1)

ABFChannelGetCount session, channel_count

For channel = 0 To channel_count-1
```

```
  ABFEpisodeRead session, channel, 0, episode_size,
    buffer(0)
  ' Data processing goes here
Next channel
```

Reading Sections of an Episode

The following example reads the first “length” values of the first episode into an array.

```
Dim buffer() As Double
Dim episode_size As Long

ABFEpisodeGetSampleCount session, episode_size

If episode_size < length Then
  ' Limit the read to the actual length of the episode
  length = episode_size
End If

ReDim buffer(length-1)

ABFEpisodeRead
  session, channel, 0, length, buffer
```

3.2.7 Closing a File

To close a file, use `ABFFileClose`. [↗ ABFFileClose, p. 73.](#)

The following example closes an ABF file, then closes the associated session.

```
ABFFileClose session
ABFSessionClose session
```

3.3 Robust Processing

If you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

All operations return a status value of type `Long`. The value is zero if the operation was successful, and non-zero if an error was detected.

If an error is detected, you can use `ABFGetStatusText` to translate the status code to a message. [↗ ABFGetStatusText, p. 82.](#)

The following example shows how the user might be allowed to select a file. The example does not terminate until the user selects a valid ABF file.

```
Dim filespec As Variant
Dim status As Long
Dim session As Long

ABFSessionOpen session

Do
    ' Add code here to obtain the file specification in "filespec"
    status = ABFFileOpen(session, filespec)

    If status = 0 Then
        Exit Do
    End If

    ' Translate the status code to a message here and
    ' display it for the user
Loop
```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

3.4 Operations

This chapter describes each of the operations provided by DataAccess. The operations are grouped by category.

3.4.1 Common Operations

The common operations are valid regardless of whether a session is open or not. The common operations are shown in table 32.

Table 32 Common operations

Operation	
ABFGetVersion	p. 83
ABFGetStatusText	p. 82

3.4.2 Session Operations

The session operations allow you to open and close a ses-

sion. The session operations are shown in table 33.

Table 33 Session Operations

Operation	
ABFSessionOpen	p. 91
ABFSessionClose	p. 90

3.4.3 File Operations

The file operations allow you to access an ABF data file. The file operations are shown in table 34.

Table 34 File operations

Operation	
ABFFileOpen	p. 80
ABFFileClose	p. 73

The file operations are valid when a session is open.

3.4.4 File Parameter Operations

The file parameter operations allow you to obtain file parameters. The file parameter operations are shown in table 35.

Table 35 File parameter operations

Operation	
ABFFileGetComment	p. 73
ABFFileGetCreator	p. 74
ABFFileGetDigital	p. 74
ABFFileGetExperimentType	p. 75
ABFFileGetIdentifier	p. 75
ABFFileGetOperationMode	p. 76
ABFFileGetSampleInterval	p. 76
ABFFileGetSecondInterval	p. 77
ABFFileGetSignature	p. 78
ABFFileGetTime	p. 78
ABFFileGetTrigger	p. 79
ABFFileGetVersion	p. 80
ABFFileParametersGet	p. 81
ABFFileParametersRead	p. 82

The file parameter operations are valid when a file is open.

3.4.5 Episode Operations

The episode operations allow you to access each episode of a file. The episode operations are shown in table 36.

Table 36 Episode operations

Operation	
ABFEpisodeGetCount	p. 67
ABFEpisodeSet	p. 70
ABFEpisodeGetDuration	p. 67
ABFEpisodeGetSampleCount	p. 68
ABFEpisodeGetStart	p. 68
ABFEpisodeRead	p. 69
ABFEpisodeReadStimulus	p. 70

The episode operations are valid when a file is open. Most require that a current episode be set.

3.4.6 Stimulus Operations

The stimulus operations allow you to determine the output data source and the data used for output. The stimulus operations are shown in table 37.

Table 37 Stimulus operations

Operation	
ABFStimulusGetName	p. 91
ABFStimulusGetRange	p. 92
ABFStimulusGetSource	p. 92
ABFStimulusGetUnits	p. 93

If the epoch parameters are used to generate a stimulus, you can use the epoch operations to determine the epoch parameters. *Epoch Operations, p. 58.*

The stimulus operations are valid when a file is open. Most require that a current episode be set.

3.4.7 Holding Potential Operations

The holding potential operations allow you to determine the amplitude and duration of the holding potential. The

holding potential operations are shown in table 38.

Table 38 Holding potential operations

Operation	
ABFHoldingInitialGetAmplitude	p. 84
ABFHoldingInitialGetDuration	p. 85
ABFHoldingFinalGetAmplitude	p. 83
ABFHoldingFinalGetDuration	p. 84

The holding potential operations are valid when a file is open.

3.4.8 Channel Operations

The channel operations allow you to determine the number of channels in a file, and obtain channel parameters. The channel operations are shown in table 39.

Table 39 Channel operations

Operation	
ABFChannelGetCount	p. 61
ABFChannelGetADC	p. 61
ABFChannelGetFilterLow	p. 62
ABFChannelGetFilterHigh	p. 62
ABFChannelGetName	p. 63
ABFChannelGetRange	p. 63
ABFChannelGetSignalConditioner	p. 64
ABFChannelGetUnits	p. 64

The channel operations are valid when a file is open.

3.4.9 Epoch Operations

The epoch operations allow you to determine the parameters used for stimulation. The epoch operations are shown in table 40.

Table 40 Epoch operations

Operation	
ABFEpochGet	p. 71
ABFEpochGetPrototype	p. 72

The epoch operations are valid when a file is open. ABFEpochGet requires that a current episode be set.

3.4.10 Leak Operations

The leak operations allow you to determine the leak subtraction parameters.

The leak operations are shown in table 41.

Table 41 Leak operations

Operation	
ABFLeakGetCount	p. 85
ABFLeakGetEnabled	p. 86
ABFLeakGetHoldingLevel	p. 86
ABFLeakGetPosition	p. 87
ABFLeakGetPolarity	p. 87
ABFLeakGetPulseInterval	p. 88
ABFLeakGetSettlingTime	p. 88

3.4.11 Conditioning Pulse Train (CPT) Operations

The conditioning pulse train (CPT) operations allow you to determine the parameters of the conditioning pulse train, if any, used during acquisition.

The conditioning pulse train operations are shown in table 42.

Table 42 Conditioning Pulse Train operations

Operation	
ABFCPTGetCount	p. 65
ABFCPTGetBaseline	p. 65
ABFCPTGetPostTrain	p. 66
ABFCPTGetStep	p. 66

The conditioning pulse train operations are valid when a file is open.

3.4.12 Run Operations

The run operations allow you to determine the run averaging used to record an episode.

The run operations are shown in table 43.

Table 43 Run operations

Operation	
ABFRunGetAlgorithm	p. 89
ABFRunGetCount	p. 89
ABFRunGetInterval	p. 90

3.4.13 Tag Operations

The tag operations allow you to read the tags stored in a data file. The tag operations are shown in table 44.

Table 44 Tag operations

Operation	
ABFTagGetComment	p. 93
ABFTagGetCount	p. 94
ABFTagGetTime	p. 94
ABFTagGetType	p. 95
ABFTagGetVoiceNumber	p. 95

The tag operations are valid when a file is open.

3.4.14 Annotation Operations

The annotation operations allow you to read the annotations stored in a data file. The annotation operations are shown in table 45. The operations are valid when a file is open.

Table 45 Annotation operations

Operation	
ABFAnnotationGetCount	p. 60
ABFAnnotationGetText	p. 60

3.5 Reference

This section lists each DataAccess operation in alphabetical order.

3.5.1 ABFAnnotationGetCount

ABFAnnotationGetCount returns the number of annotations.

Syntax

ABFAnnotationGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of annotations.

Example

```
Dim count As Long
```

```
ABFAnnotationGetCount session, count
```

Discussion

You must have a file open.

3.5.2 ABFAnnotationGetText

ABFAnnotationGetText returns the text associated with an annotation.

Syntax

ABFAnnotationGetText *session, annotation, text*

session A Long value representing the current session.

annotation A Long value representing the annotation number to return.

comment A Variant variable in which the text is returned.

Example

```
Dim text As Variant
```

```
ABFAnnotationGetText session, 0, text
```

Discussion

You must have a file open.

Annotations are numbered 0 to N-1, where N is the value returned by ABFAnnotationGetCount. [↗ ABFAnnotationGetCount, p. 60.](#)

3.5.3 ABFChannelGetADC

ABFChannelGetADC returns the device channel number of the specified channel.

Syntax

ABFChannelGetADC *session, channel, adc*

session A Long value representing the current session.

channel A Long value representing the channel to use.

adc A Long variable to receive the channel number.

Example

```
Dim adc As Long
```

```
ABFChannelGetADC session, 1, adc
```

Discussion

You must have a file open.

Channel numbers are logical, and may not correspond to the channel number of a device. The adc value returned is the channel number on the front panel of the data acquisition device.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount.  *ABFChannelGetCount*, p. 61.

3.5.4 ABFChannelGetCount

ABFChannelGetCount returns the number of channels recorded in the current ABF data file.

Syntax

ABFChannelGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of channels.

Example

```
Dim channel_count As Long
```

```
ABFChannelGetCount session, channel_count
```

Discussion

You must have a file open.

The channel count is the number of active channels recorded in the ABF data file. For example, if data was recorded from channels 0, 2, and 5, the channel count is 3. The channel count must be at least 1.

3.5.5 ABFChannelGetFilterHigh

ABFChannelGetFilterHigh returns information about the high pass filter used.

Syntax

ABFChannelGetFilterHigh *session, channel, type, cutoff*

session A Long value representing the current session.

channel A Long value representing the channel to use.

type A Long variable to receive the high pass filter type.

cutoff A Double variable to receive the filter cutoff frequency.

Example

```
Dim type As Long
Dim cutoff As Double
```

```
ABFChannelGetFilterHigh session, type, cutoff
```

Discussion

You must have a file open.

The cutoff frequency is measured in Hz. If the cutoff frequency is 0, the filter is bypassed. If the cutoff frequency is -1, the inputs are grounded.

The filter type is one of the values shown in table 46.

Table 46 Filter types

Value	Type
0	None
1	External
2	Simple
3	Bessel
4	Butterworth

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [☞ ABFChannelGetCount, p. 61.](#)

3.5.6 ABFChannelGetFilterLow

ABFChannelGetFilterLow returns information about the low pass filter used.

Syntax

ABFChannelGetFilterLow *session, channel, type, cutoff*

session A Long value representing the current session.

channel A Long value representing the channel to use.

type A Long variable to receive the low pass filter type.

cutoff A Double variable to receive the filter cutoff frequency.

Example

```
Dim type As Long
Dim cutoff As Double
```

```
ABFChannelGetFilterLow session, type, cutoff
```

Discussion

You must have a file open.

The cutoff frequency is measured in Hz. If the frequency is 100000, it means that filtering is bypassed.

The filter type is one of the values shown in table 46.

Table 47 Filter types

Value	Type
0	None
1	External
2	Simple
3	Bessel
4	Butterworth

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [☞ ABFChannelGetCount, p. 61.](#)

3.5.7 ABFChannelGetName

ABFChannelGetName returns the name of the specified channel.

Syntax

ABFChannelGetName *session, channel, name*

session A Long value representing the current session.

channel A Long value representing the channel to use.

name A Variant variable in which the channel name is returned.

Example

Dim text As Variant

ABFChannelGetName session, 0, text

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [↗ ABFChannelGetCount, p. 61.](#)

3.5.8 ABFChannelGetRange

ABFChannelGetRange returns the range of the specified channel.

Syntax

ABFChannelGetRange *session, channel, minimum, maximum*

session A Long value representing the current session.

channel A Long value representing the channel to use.

minimum A Double variable to receive the lower limit.

maximum A Double variable to receive the upper limit.

Example

Dim minimum As Double

Dim maximum As Double

ABFChannelGetRange session, 0, minimum, maximum

Discussion

You must have a file open.

ABFChannelGetRange returns the minimum and maximum possible values for the specified channel.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [↗ ABFChannelGetCount, p. 61.](#)

3.5.9 ABFChannelGetSignalConditioner

ABFChannelGetSignalConditioner returns the signal conditioner for the specified channel.

Syntax

ABFChannelGetSignalConditioner *session, channel, conditioner*

session A Long value representing the current session.

channel A Long value representing the channel to use.

conditioner A Long variable in which to store the signal conditioner.

Example

```
Dim channel_conditioner As Long
```

```
ABFChannelGetSignalConditioner session, 0,  
channel_conditioner
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [↗ ABFChannelGetCount, p. 61.](#)

The conditioner is one of the values shown in table 48.

Table 48 Conditioner values

Value	Conditioner
0	None
1	CYBERAMP 320380

3.5.10 ABFChannelGetUnits

ABFChannelGetUnits returns the units of the specified channel.

Syntax

ABFChannelGetUnits *session, channel, units*

session A Long value representing the current session.

channel A Long value representing the channel to use.

units A Variant variable in which the channel units are returned.

Example

```
Dim text As Variant
```

```
ABFChannelGetUnits session, 0, text
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [↗ ABFChannelGetCount, p. 61.](#)

3.5.11 ABFCPTGetBaseline

ABFCPTGetBaseline returns conditioning pulse train pulse baseline information.

Syntax

ABFCPTGetBaseline *session, channel, duration, amplitude*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

duration A Double variable in which to store the baseline duration.

amplitude A Double variable in which to store the baseline amplitude.

Example

```
Dim duration As Double
Dim amplitude As Double
```

```
ABFCPTGetBaseline session, 0, duration, amplitude
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

A conditioning pulse train pulse consists of a baseline segment and a step segment. ABFCPTGetStep returns information about the step segment. [↗ ABFCPTGetStep, p. 66.](#)

3.5.12 ABFCPTGetCount

ABFCPTGetCount returns the number of conditioning pulse train pulses.

Syntax

ABFCPTGetCount *session, channel, count*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

count A Long variable in which to store the number of pulses.

Example

```
Dim pulse_count As Long
```

```
ABFCPTGetCount session, 0, pulse_count
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

3.5.13 ABFCPTGetPostTrain

ABFCPTGetPostTrain returns post conditioning pulse train information.

Syntax

ABFCPTGetPostTrain *session, channel, duration, amplitude*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

duration A Double variable in which to store the post train duration.

amplitude A Double variable in which to store the post train amplitude.

Example

```
Dim post_duration As Double
Dim psot_amplitude As Double
```

```
ABFCPTGetPostTrain session, 0, duration, amplitude
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

At the end of the conditioning pulse train there is a steady-state output. ABFCPTGetPostTrain describes this output.

3.5.14 ABFCPTGetStep

ABFCPTGetStep returns conditioning pulse train pulse step information.

Syntax

ABFCPTGetStep *session, channel, duration, amplitude*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

duration A Double variable in which to store the step duration.

amplitude A Double variable in which to store the step amplitude.

Example

```
Dim pulse_duration As Double
Dim pulse_amplitude As Double
```

```
ABFCPTGetStep session, 0, duration, amplitude
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

A conditioning pulse train pulse consists of a baseline segment and a step segment. ABFCPTGetBaseline returns information about the baseline segment. [☞ ABFCPTGetBaseline, p. 65.](#)

3.5.15 ABFEpisodeGetCount

ABFEpisodeGetCount returns the number of episodes recorded in the file.

Syntax

ABFEpisodeGetCount *session, count*

session A Long value representing the current session.

count A Long variable in which to store the number of episodes.

Example

```
Dim episode_count As Long
```

```
ABFEpisodeGetCount session, episode_count
```

Discussion

You must have a file open.

The episode count is the number of episodes recorded in the ABF data file. Episodes are numbered 1 to N, where N is the number of episodes in the file.

3.5.16 ABFEpisodeGetDuration

ABFEpisodeGetDuration returns the duration of the current episode.

Syntax

ABFEpisodeGetDuration *session, duration*

session A Long value representing the current session.

duration A Double variable in which to return the episode duration.

Example

```
Dim duration As Double
```

```
ABFEpisodeSet session, episode
ABFEpisodeGetDuration session, duration
```

Discussion

You must have a file open and an episode selected.
 ➤ *ABFEpisodeSet*, p. 70.

The duration of an episode is measured in seconds.

3.5.17 ABFEpisodeGetSampleCount

ABFEpisodeGetSampleCount returns the number of samples in the current episode.

Syntax

ABFEpisodeGetSampleCount *session, samples*

session A Long value representing the current session.

samples A Long variable in which to return the number of samples in the episode.

Example

Dim samples As Long

ABFEpisodeGetSampleCount session, samples

Discussion

You must have a file open and an episode selected.
 ☞ *ABFEpisodeSet*, p. 70.

The number of samples in an episode is measured per channel. All channels have the same number of samples.

3.5.18 ABFEpisodeGetStart

ABFEpisodeGetStart returns the starting time of the current episode.

Syntax

ABFEpisodeGetStart *session, start*

session A Long value representing the current session.

start A Double variable in which to return the episode starting time.

Example

Dim start As Double

ABFEpisodeSet session, episode
 ABFEpisodeGetDuration session, start

Discussion

You must have a file open and an episode selected.
 ☞ *ABFEpisodeSet*, p. 70.

The start time of an episode is measured in seconds since the beginning of the experiment.

3.5.19 ABFEpisodeRead

ABFEpisodeRead reads the data for one channel of a section of the current episode.

Syntax

```
ABFEpisodeRead
    session, channel, start, length, data
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

start A Long value representing the first sample to read.

length A Long value representing the number of samples to read.

data An array of “Double” to receive the data. The array must be long enough to contain all the data in the section.

Example

```
Dim samples As Long
Dim start As Long
Dim buffer() As Double
```

```
ABFEpisodeGetSampleCount session, samples
```

```
samples = samples/2
start = samples/2
```

```
ReDim buffer(samples)
```

```
ABFEpisodeRead
    session, channel, start, samples, buffer(0)
```

Discussion

You must have a file open and an episode selected. [↗ ABFEpisodeSet, p. 70.](#)

The start and length values must satisfy the following criteria:

- 1 The start value must be positive.
- 2 The length value must be greater than zero.

- 3 The sum of the start and length values must be less than the episode length.

To ensure that the parameters are in range, you can use ABFEpisodeGetSampleCount to determine the number of data values in the episode. [↗ ABFEpisodeGetSampleCount, p. 68.](#)

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use ABFChannelGetCount. [↗ ABFChannelGetCount, p. 61.](#)

3.5.20 ABFEpisodeReadStimulus

ABFEpisodeReadStimulus reads the output data for the current episode into an array of Double values.

Syntax

ABFEpisodeReadStimulus *session, channel, data*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

data An array of “Double” to receive the data. The array must be long enough to contain the data in the episode.

Example

```
Dim samples As Long
Dim buffer() As Double
```

```
ABFEpisodeGetSampleCount session, samples
ReDim buffer(samples)
```

```
ABFEpisodeReadStimulus session, 0, buffer(0)
```

Discussion

You must have a file open and an episode selected. [☞ ABFEpisodeSet, p. 70.](#)

The stimulus channels are numbered 0 to 1.

To ensure that the array is long enough, you can use ABFEpisodeGetSampleCount to determine the number of data values in the episode. [☞ ABFEpisodeGetSampleCount, p. 68.](#)

The output data may be generated from the epoch information or provided as a sequence of samples (DAC file). In either case ABFEpisodeReadStimulus will provide the output data. To determine the source of the data, use ABFStimulusGetSource. [☞ ABFStimulusGetSource, p. 92.](#)

The returned data are in the user units. [☞ ABFStimulusGetUnits, p. 93.](#)

3.5.21 ABFEpisodeSet

ABFEpisodeSet selects the current episode.

Syntax

ABFEpisodeSet *session, episode*

session A Long value representing the current session.

episode A Long value representing the episode.

Example

```
Dim episodes As Long
```

```
ABFEpisodeGetCount session, episodes
```

```
ABFEpisodeGetDuration session, episodes, duration
```

Discussion

You must have a file open.

Episodes are numbered 1 to N, where N is the number of episodes in the file. [☞ ABFEpisodeGetCount, p. 67.](#)

3.5.22 ABFEpochGet

ABFEpochGet returns the parameters for the specified epoch for the current episode.

Syntax

ABFEpochGetAmplitude *channel, session, epoch, type, duration, amplitude, digital*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

epoch A Long value representing the epoch.

type A Long variable to receive the type of the epoch.

duration A Long variable to receive the duration of the epoch in samples.

amplitude A Double variable to receive the amplitude of the epoch.

digital A Long variable to receive the digital outputs for the epoch.

Example

```
Dim type As Long
Dim duration As Double
Dim amplitude As Double
Dim digital As Long
```

```
ABFEpochGetAmplitude session, 0, 0, type, duration,
    amplitude, digital
```

Discussion

You must have a file open and an episode selected.
☞ ABFEpisodeSet, p. 70.

The stimulus channels are numbered 0 to 1.

Type values are shown in table 49.

Table 49 Epoch types

Value	Type
0	Disabled
1	Step
2	Ramp

The type of an epoch is fixed for all episodes.

The amplitude, duration and digital output used for an epoch may vary with each episode. ABFEpochGet returns the values for the current episode.

The duration is the number of samples output during the epoch.

The amplitude is in the units returned by ABFStimulusGetUnits. ☞ ABFStimulusGetUnits, p. 93.

The digital value represents the digital output on/off states for the epoch.

Epochs are numbered 0 to 9.

3.5.23 ABFEpochGetPrototype

ABFEpochGetPrototype returns the information from which the epoch data is constructed.

Syntax

ABFEpochGetPrototype *session, channel, epoch, type, amplitude, amplitude_increment, duration, duration_increment*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

epoch A Long value representing the epoch.

type A Long variable to receive the type of the epoch.

amplitude A Double variable to receive the base amplitude of the epoch.

amplitude_increment A Double variable to receive the amplitude increment of the epoch.

duration A Double variable to receive the base duration of the epoch.

duration_increment A Double variable to receive the duration increment of the epoch.

PulsePeriod A Long variable to receive the pulse period for a pulse train epoch.

PulseWidth A Long variable to receive the pulse width for a pulse train epoch.

Example

```
Dim amplitude As Double
Dim amplitude_increment As Double
Dim duration As Double
Dim duration_increment As Double
Dim actual_amplitude As Double
Dim actual_duration As Double
Dim PulsePeriod As Long
Dim PulseWidth As Long
```

```
ABFEpochGetPrototype session, 0, 0, amplitude,
amplitude_increment, duration, duration_increment,
```

```
PulsePeriod, PulseWidth
```

```
actual_amplitude =
amplitude + (episode - 1) * amplitude_increment
```

```
actual_duration =
duration + (episode - 1) * duration_increment
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The duration of an epoch is measured in seconds.

When epochs are used to generate a stimulus, the actual epoch amplitude and duration are calculated using the equations shown in the example. That is, during the first episode, the base amplitude and base duration are used. During the second episode, the base amplitude plus the amplitude increment, and the base duration plus the duration increment are used.

The stimulus amplitude and duration might not match the parameters returned by ABFEpochGetPrototype in several cases:

- 1 The user list can be used to specify a set of values for an epoch amplitude or duration. The user list overrides the epoch prototype.
- 2 The file might use a DAC file instead of epochs to generate output. This can be determined using ABFStimulusGetSource. [☞ ABFStimulusGetSource, p. 92.](#)

The epoch prototype is independent of the episode.

The amplitudes are in user units. [☞ ABFStimulusGetUnits, p. 93.](#)

Epochs are numbered 0 to 9.

Type values are shown in table 49.

3.5.24 ABFFileClose

ABFFileClose terminates processing of an ABF file.

Syntax

```
ABFFileClose session
```

session A Long value representing the current session.

Example

```
ABFFileClose session  
ABFSessionClose session
```

Discussion

You must have a file open.

You should perform ABFFileClose as soon as you complete processing of a file.

3.5.25 ABFFileGetComment

ABFFileGetComment returns the comment for the file.

Syntax

```
ABFFileGetComment session, comment
```

session A Long value representing the current session.

comment A Variant variable in which the file comment is returned.

Example

```
Dim text As Variant  
  
ABFGetFileComment session, text
```

Discussion

You must have a file open.

3.5.26 ABFFileGetCreator

ABFFileGetCreator returns the creator for the file.

Syntax

ABFFileGetCreator *session, creator*

session A Long value representing the current session.

creator A Variant variable in which the file creator is returned.

Example

Dim text As Variant

ABFFileGetCreator session, text

Discussion

You must have a file open.

3.5.27 ABFFileGetDigital

ABFFileGetDigital returns the digital output settings for the file.

Syntax

ABFFileGetDigital *session, enabled, holding*

session A Long value representing the current session.

enabled A Long variable to receive the enabled state of the digital outputs.

holding A Long variable to receive the digital holding values.

Example

Dim enabled As Long

Dim holding As Long

ABFFileGetDigital session, enabled, holding

Discussion

You must have a file open.

3.5.28 ABFFileGetExperimentType

ABFFileGetExperimentType returns the type of experiment recorded in the file.

Syntax

ABFFileGetExperimentType *session, type*

session A Long value representing the current session.

type A Long variable to receive the experiment type.

Example

```
Dim experiment_type As Long
```

```
ABFFileGetExperimentType session, experiment_type
```

Discussion

You must have a file open.

The experiment types are shown in table 50.

Table 50 Experiment types

Value	Type
0	Voltage clamp
1	Current clamp

3.5.29 ABFFileGetIdentifier

ABFFileGetIdentifier returns cell identification values.

Syntax

ABFFileGetIdentifier *session, identifier, value*

session A Long value representing the current session.

identifier An Long value containing the identifier number.

value A Double variable to receive the identifier value.

Example

```
Dim value As Double
```

```
ABFFileGetIdentifier session, 1, value
```

Discussion

Identifiers are numbered 1 to 3. The identifier values are entered by the user.

3.5.30 ABFFileGetOperationMode

ABFFileGetOperationMode returns the mode used to record data in the file.

Syntax

ABFFileGetOperationMode *session, mode*

session A Long value representing the current session.

mode A Long variable to receive the operation mode.

Example

```
Dim operation_mode As Long
ABFFileGetOperationMode session, operation_mode
```

Discussion

You must have a file open.

The operation mode values are shown in table 51.

Table 51 Operation mode values

Value	Mode
1	Variable-length events
2	Fixed-length events
3	Gap-free
4	High-speed oscilloscope
5	Clampex

3.5.31 ABFFileGetSampleInterval

ABFFileGetSampleInterval returns the sampling interval in seconds on each channel.

Syntax

ABFFileGetSampleInterval *session, interval*

session A Long value representing the current session.

interval A Double variable to receive the sampling interval.

Example

```
Dim interval As Double
ABFFileGetSampleInterval session, interval
```

Discussion

You must have a file open.

All channels are sampled at the same interval. The interval measures the time between samples on the same channel.

The sampling interval is the same throughout the file, unless a second sampling interval is specified. If a second sampling interval is specified, the sampling interval may change during each episode. [↗ ABFFileGetSecondInterval, p. 77.](#)

The sampling interval is reported in units of seconds. It always represents an integer number of microseconds. To convert the sampling interval to the exact number of microseconds, use code similar to the following:

```
Dim interval As Double
Dim microseconds As Long

ABFFileGetSampleInterval session, interval

microseconds = CInt(interval * 1000000.0)
```

Notice that the conversion from “Double” to “Long” is handled using “CInt”, which rounds to the nearest integer value. This compensates for potential roundoff error in the representation of the sampling interval.

3.5.32 ABFFileGetSecondInterval

ABFFileGetSecondInterval returns a description of the second sampling interval, including the sample interval and the sample number in each episode from which sampling is performed using the second sampling interval.

Syntax

```
ABFFileGetSecondIntervalStart session, interval, sample
```

session A Long value representing the current session.

interval A Double variable to receive the second sampling interval.

sample A Long variable to receive the sample number from which sampling is performed using the second sampling interval.

Example

```
Dim interval As Double
Dim sample As Long
```

```
ABFFileGetSecondInterval session, interval, sample
```

```
If interval <> 0.0 Then
    ' The second sampling interval is used
End If
```

Discussion

You must have a file open.

All channels are sampled at the same interval. The interval measures the time between samples on the same channel.

If the second sampling interval is non-zero, the sample number specifies the sample number within each episode from which the second sampling interval is used. Samples preceding the sample number are sampled using the normal sampling interval. ↗ *ABFFileGetSampleInterval*, p. 76.

The second sampling interval is reported in units of seconds. It always represents an integer number of microseconds. To convert the second sampling interval to the exact

number of microseconds, use code similar to the following:

```
Dim interval As Double
Dim microseconds As Long
Dim sample As Long
```

```
ABFFileGetSecondInterval session, interval, sample
```

```
microseconds = CInt(interval * 1000000.0)
```

Notice that the conversion from “Double” to “Long” is handled using “CInt”, which rounds to the nearest integer value. This compensates for potential roundoff error in the representation of the sampling interval.

3.5.33 ABFFileGetSignature

ABFFileGetSignature returns a signature that uniquely identifies the file.

Syntax

```
ABFFileGetSignature session, signature
```

session A Long value representing the current session.

signature A Variant variable in which the file signature is returned.

Example

```
Dim signature As Variant
```

```
ABFFileGetSignature session, signature
```

Discussion

You must have a file open.

The returned signature is a string containing a 32 character hexadecimal value.

The signature is calculated from information in the file. Except for a very unlikely coincidence, the signature should uniquely identify the file.

3.5.34 ABFFileGetTime

ABFFileGetTime returns the creation date and time of the file.

Syntax

```
ABFFileGetTime session, date_time
```

session A Long value representing the current session.

date_time A Date variable to receive the file creation date and time.

Example

```
Dim date_time As Date
```

```
ABFFileGetTime session, date_time
```

Discussion

You must have a file open.

3.5.35 ABFFileGetTrigger

ABFFileGetTrigger returns the trigger parameters.

Syntax

```
ABFFileGetTrigger session, action, interval, polarity,
source, threshold
```

session A Long value representing the current session.

action A Long variable to receive the trigger action.

interval A Double variable to receive the trigger episode interval.

polarity A Long variable to receive the trigger polarity.

source A Long variable to receive the trigger source.

threshold A Double variable to receive the trigger threshold.

Example

```
Dim trigger_action As Long
Dim trigger_interval As Double
Dim trigger_polarity As Long
Dim trigger_source As Long
Dim trigger_threshold As Double
```

```
ABFFileGetTrigger session, trigger_action,
trigger_interval, trigger_polarity, trigger_source,
trigger_threshold
```

Discussion

You must have a file open.

The trigger action is one of the values in table 52.

Table 52 Trigger actions

Value	Action
0	Start one sweep
1	Start one run
2	Start one trial

The trigger source is one of the values in table 53.

Table 53 Trigger sources

Value	Trigger source
0 or positive	channel number
-1	external
-2	keyboard
-3	start to start interval

The episode interval is the time between episode starts. The episode interval is used only when the trigger source is 'start to start interval'.

3.5.36 ABFFileGetVersion

ABFFileGetVersion returns the ABF format version of the file.

Syntax

```
ABFFileGetVersion session, version
```

session A Long value representing the current session.

version A Variant variable in which the file version is returned.

Example

```
Dim version As Variant
```

```
ABFFileGetVersion session, version
```

Discussion

You must have a file open.

3.5.37 ABFFileOpen

ABFFileOpen prepares an ABF data file for processing.

Syntax

```
ABFFileOpen session, path
```

session A Long value representing the current session.

path A Variant value representing the file path.

Example

```
Dim file As Variant
```

```
file = "Data.dat"
```

```
ABFFileOpen session, file
```

Discussion

You must not have a file open.

ABFFileOpen loads the ABF file header into memory. This header describes all the parameters of the file.

Once a file is open, you can access file parameters. You can process the episodes in the file. [Episode Operations, p. 58.](#)
[File Parameter Operations, p. 57.](#)

When you are done with a file, you should close it. [ABFFileClose, p. 73.](#)

ABFFileOpen invalidates the current episode. You must set the current episode using ABFEpisodeSet. [ABFEpisodeSet, p. 70.](#)

3.5.38 ABFFileParametersGet

ABFFileParametersGet returns information about the user parameter list.

Syntax

```
ABFFileParametersGet session, index, enabled, parameter,
count, repeat
```

session A Long value representing the current session.

index An int value indicating the parameter index.

enabled A Long variable to receive the enabled state of the user parameter list.

parameter A Long variable to receive the value indicating which parameter is varied.

count A Long variable to receive number of values contained in the list.

repeat A Long variable to receive the repeat mode.

Example

```
Dim enabled As Long
Dim paramter As Long
Dim count As Long
Dim repeat As Long
```

```
ABFFileParametersGet session, enabled, parameter,
count, repeat
```

Discussion

You must have a file open.

Index can range from 0 to 3.

A parameter that varies from episode to episode may be overridden by values stored in the parameter list.

If the value returned for enabled is zero, no parameter value is returned. If the value returned for enabled is non-zero, a parameter value is returned. The possible parameter

values are listed in table 54.

Table 54 Parameters

Value	Varying parameter
0	number of conditioning pules
1	conditioning baseline duration
2	conditioning baseline level
3	conditioning step duration
4	conditioning step level
5	conditioning post train duration
6	conditioning post train level
7	episode start to start
8	inactive holding level
9	digital inter episode
10	number of P/N pulses
11-20	epoch digital value, epoch number value-11
21-30	epoch level, epoch number value-21
31-40	epoch duration, epoch number value-31
41-50	pulse train period, epoch number value-41
51-60	pulse train width, epoch number value-51

If count is less than the number of episodes in the file, and repeat is zero the last value in the list is used for the remainder of the episodes. Otherwise the list is repeated.

The parameter list is stored in the file as text, and can possibly contain invalid entries. In this case the count value returned is the number of valid entries. If the count value is zero, no entries are valid, and the parameter list is ignored.

Use ABFFileParametersRead to obtain the values in the parameter list. [↗ ABFFileParametersRead, p. 82.](#)

3.5.39 ABFFileParametersRead

ABFFileParametersRead returns the values in the user parameter list.

Syntax

```
ABFFileParametersRead session, index, values, count
```

session A Long value representing the current session.

index An int value indicating the parameter index.

values An array of Double variables to receive the parameter list values.

count A Long value indicating the number of parameter list values to read.

Example

```
Dim buffer() As Double
Dim episode_count As Long

ABFEpisodeGetCount session, episode_count

ReDim buffer(episode_count)

ABFFileParametersRead session, buffer, episode_count
```

Discussion

You must not have a file open.

Index can range from 0 to 3.

If count is larger than the number of values in the parameter list in the file, the additional entries in the parameters array are filled with the last value in the parameter list.

If you call ABFFileParametersRead with count equal to the number of episodes, the parameter list will be filled with the values used for each episode. Use ABFEpisodeGetCount to determine the number of episodes.

☞ *ABFEpisodeGetCount*, p. 67.

3.5.40 ABFGetStatusText

ABFGetStatusText translates a status value returned by a DataAccess operation to a text string.

Syntax

```
ABFGetStatusText session, status, message
```

session A Long value representing the current session.

status A Long value representing the status value to translate.

message A Variant variable in which the translated text is returned.

Example

```
Dim status As Long
Dim message As Variant
Dim file_name As Variant
file_name = "Sample.dat"

status = ABFFileOpen(session, file_name)

If status <> 0 Then
    ABFGetStatusText session, status, message
    ' Display the message here
End If
```

Discussion

ABFGetStatusText does not return a value, so it should not be called as a function.

You need not have a file open.

ABFGetStatusText is the only means provided to interpret status values.

3.5.41 ABFGetVersion

ABFGetVersion returns the version number of the DataAccess package.

Syntax

ABFGetVersion *version*

version A Long variable to receive the DataAccess version number.

Example

```
Dim version As Long
ABFGetVersion version
```

Discussion

ABFGetVersion does not return a value, so it should not be called as a function.

You need not have either a session or a file open.

If you are writing a set of procedures using DataAccess, you can use ABFGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess. You can use the following code to ensure that this version or a later version is in use:

```
Dim version As Long
ABFGetVersion version

If version < N Then
    ' Handle the error
End If
```

3.5.42 ABFHoldingFinalGetAmplitude

ABFHoldingFinalGetAmplitude returns the final holding amplitude of each episode.

Syntax

ABFHoldingFinalGetAmplitude *session, channel, amplitude*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

amplitude A Double variable to receive the holding amplitude.

Example

```
Dim amplitude As Double
ABFHoldingFinalGetAmplitude session, 0, amplitude
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The holding amplitude is the same for all episodes.

The amplitude is in user units. ➔ *ABFStimulusGetUnits*, p. 93.

3.5.43 ABFHoldingFinalGetDuration

ABFHoldingFinalGetDuration returns the final holding duration of each episode.

Syntax

ABFHoldingFinalGetDuration *session, channel, duration*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

duration A Double variable to receive the holding duration.

Example

```
Dim duration As Double
```

```
ABFHoldingFinalGetDuration session, 0, duration
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The duration is measured in seconds.

The holding duration is the same for all episodes.

3.5.44 ABFHoldingInitialGetAmplitude

ABFHoldingInitialGetAmplitude returns the initial holding amplitude of each episode.

Syntax

ABFHoldingInitialGetAmplitude *session, channel, amplitude*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

amplitude A Double variable to receive the holding amplitude.

Example

```
Dim amplitude As Double
```

```
ABFHoldingInitialGetAmplitude session, 0, amplitude
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The holding amplitude is the same for all episodes.

The amplitude is in user units. ➔ *ABFStimulusGetUnits*, p. 93.

3.5.45 ABF HoldingInitialGetDuration

ABF HoldingInitialGetDuration returns the initial holding duration of each episode.

Syntax

```
ABF HoldingInitialGetDuration session, channel, duration
```

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

duration A Double variable to receive the holding duration.

Example

```
Dim duration As Double
```

```
ABF HoldingInitialGetDuration session, 0, duration
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The duration is measured in seconds.

The holding duration is the same for all episodes.

3.5.46 ABF LeakGetCount

ABF LeakGetCount returns the number of P/N subpulses.

Syntax

```
ABF LeakGetCount session, channel, count
```

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

count A Long variable in which to store the number of subpulses.

Example

```
Dim leak_count As Long
```

```
ABF LeakGetCount session, 0, leak_count
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

3.5.47 ABFLeakGetEnabled

ABFLeakGetEnabled returns the P/N subtraction usage.

Syntax

ABFLeakGetEnabled *session, channel, enabled*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

enabled A Long variable in which to store the P/N subtraction usage.

Example

```
Dim leak_enabled As Long
```

```
ABFLeakGetEnabled session, 0, leak_enabled
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

ABFLeakGetEnabled returns 0 if P/N subtraction is disabled and 1 if P/N subtraction is enabled.

3.5.48 ABFLeakGetHoldingLevel

ABFLeakGetHoldingLevel returns the P/N holding level.

Syntax

ABFLeakGetHoldingLevel *session, channel, level*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

level A Double variable in which to store the P/N holding level.

Example

```
Dim leak_holding_level As Long
```

```
ABFLeakGetHoldingLevel session, 0,  
leak_holding_level
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The holding level is the output level between subpulses.

3.5.49 ABFLeakGetPosition

ABFLeakGetPosition returns the P/N position.

Syntax

ABFLeakGetPosition *session, position*

session A Long value representing the current session.

position A Double variable in which to store the P/N position.

Example

```
Dim leak_position As Long
```

```
ABFLeakGetPosition session, leak_position
```

Discussion

You must have a file open.

The possible values of the leak position are shown in table 55.

Table 55 Leak position values

Value	Position
0	Leak subpulses occur before main stimulus
1	Leak subpulses occur after main stimulus

3.5.50 ABFLeakGetPolarity

ABFLeakGetPolarity returns the P/N polarity.

Syntax

ABFLeakGetPolarity *session, channel, polarity*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

polarity A Long variable in which to store the P/N polarity.

Example

```
Dim leak_polarity As Long
```

```
ABFLeakGetPolarity session, 0, leak_polarity
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The possible values of the leak polarity are shown in table 56.

Table 56 Leak polarity values

Value	Polarity
-1	Leak subpulses have polarity opposite main stimulus
1	Leak subpulses have same polarity as main stimulus

3.5.51 ABFLeakGetPulseInterval

ABFLeakGetPulseInterval returns the P/N pulse interval.

Syntax

ABFLeakGetPulseInterval *session, interval*

session A Long value representing the current session.

interval A Long variable in which to store the P/N pulse interval.

Example

```
Dim leak_interval As Long
```

```
ABFLeakGetPulseInterval session, leak_interval
```

Discussion

You must have a file open.

The interval is the time between subpulse starts in seconds.

3.5.52 ABFLeakGetSettlingTime

ABFLeakGetSettlingTime returns the P/N pulse settling time.

Syntax

ABFLeakGetSettlingTime *session, time*

session A Long value representing the current session.

time A Double variable in which to store the P/N settling time.

Example

```
Dim settling_time As Long
```

```
ABFLeakGetSettlingTime session, settling_time
```

Discussion

You must have a file open.

The settling time is measured in seconds. It is the time between the subpulses and the main stimulus.

3.5.53 ABFRunGetAlgorithm

ABFRunGetAlgorithm returns the algorithm used to average runs.

Syntax

ABFRunGetAlgorithm *session, algorithm, weighting*

session A Long value representing the current session.

algorithm A Long variable to receive the algorithm.

weighting A Double variable to receive the weighting factor.

Example

```
Dim run_algorithm As Long
Dim run_weighting As Double
```

```
ABFRunGetAlgorithm session, run_algorithm,
run_weighting
```

Discussion

You must have a file open.

The possible values of the algorithm are shown in table 57.

Table 57 Run algorithm values

Value	Algorithm
0	Cumulative averaging
1	Most recent (weighted) averaging

3.5.54 ABFRunGetCount

ABFRunGetCount returns the number of runs used to record data in the file.

Syntax

ABFRunGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the run count.

Example

```
Dim run_count As Long

ABFRunGetCount session, run_count
```

Discussion

You must have a file open.

3.5.55 ABFRunGetInterval

ABFRunGetInterval returns the interval between runs.

Syntax

ABFRunGetInterval *session, interval*

session A Long value representing the current session.

interval A Double variable to receive the run interval.

Example

```
Dim run_interval As Double
ABFRunGetInterval session, run_interval
```

Discussion

You must have a file open.

The interval is the time between run starts in seconds.

3.5.56 ABFSessionClose

ABFSessionClose ends the session. It deallocates any resources allocated DataAccess.

Syntax

ABFSessionClose *session*

session A Long value representing the current session.

Example

```
ABFFileClose session
ABFSessionClose session
```

Discussion

ABFSessionClose does not return a value, so it should not be called as a function.

You should close a session to deallocate any resources allocated to the session.

3.5.57 ABFSessionOpen

ABFSessionOpen begins a DataAccess session.

Syntax

ABFSessionOpen *session*

session A Long variable to contain the created session.

Example

```
Dim session As Long
Dim result As Long
Dim file_name As Variant
file_name = "Sample.dat"
```

```
ABFSessionOpen session
```

```
result = ABFFileOpen(session, file_name)
```

Discussion

You must have a session open to use any other DataAccess calls. The handle returned by ABFSessionOpen is a parameter to all other calls.

You can have as many sessions open simultaneously as you would like. For example, if you want to have two data files open simultaneously, call ABFSessionOpen twice. Use one handle when accessing one file and the other handle when accessing the other file.

After you are done with the session, you should call ABFSessionClose. [☞ ABFSessionClose, p. 90.](#)

3.5.58 ABFStimulusGetName

ABFStimulusGetName returns the name associated with the stimulus.

Syntax

ABFStimulusGetName *session, channel, name*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

name A Variant variable to receive the stimulus name.

Example

```
Dim name As Variant
```

```
ABFStimulusGetName session, 0, name
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

3.5.59 ABFStimulusGetRange

ABFStimulusGetRange returns the range of the stimulus.

Syntax

ABFStimulusGetRange *session, minimum, maximum*

session A Long value representing the current session.

minimum A Double variable to receive the lower limit.

maximum A Double variable to receive the upper limit.

Example

```
Dim minimum As Double
Dim maximum As Double
```

```
ABFStimulusGetRange session, minimum,
    maximum
```

Discussion

You must have a file open.

ABFStimulusGetRange returns the minimum and maximum possible values for the stimulus.

3.5.60 ABFStimulusGetSource

ABFStimulusGetSource returns the source used to construct the stimulus.

Syntax

ABFStimulusGetSource *session, channel, source*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

source A Long variable in which to return the source indicator.

Example

```
Dim stimulus_source As Long
```

```
ABFStimulusGetSource session, 0, stimulus_source
```

Discussion

You must have a file open.

The stimulus channels are numbered 0 to 1.

The source indicator values are shown in table 58.

Table 58 StimulusStimulus Source Indicator Values

Value	Meaning
0	No stimulus
1	Stimulus generated from epochs
2	Stimulus generated from DAC file

3.5.61 ABFStimulusGetUnits

ABFStimulusGetUnits returns the units associated with the stimulus.

Syntax

ABFStimulusGetUnits *session, channel, units*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

units A Variant variable in which the units of the stimulus are returned.

Example

```
Dim units As Variant
```

```
ABFStimulusGetUnits session, 0, units
```

Discussion

The stimulus channels are numbered 0 to 1.

The units are entered by the user during data acquisition.

3.5.62 ABFTagGetComment

ABFTagGetComment returns the comment associated with a tag.

Syntax

ABFTagGetComment *session, tag, comment*

session A Long value representing the current session.

tag A Long value representing the tag number to return.

comment A Variant variable in which the comment is returned.

Example

```
Dim text As Variant
```

```
ABFTagGetComment session, 0, text
```

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. [↗ ABFTagGetCount, p. 94.](#)

3.5.63 ABFTagGetCount

ABFTagGetCount returns the number of file tags.

Syntax

ABFTagGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of tags.

Example

```
Dim count As Long
```

```
ABFTagGetCount session, count
```

Discussion

You must have a file open.

3.5.64 ABFTagGetTime

ABFTagGetTime returns the time associated with a tag.

Syntax

ABFTagGetTime *session, tag, time*

session A Long value representing the current session.

tag A Long value representing the tag number to return.

time A Double variable in which the tag time is returned.

Example

```
Dim tag_time As Double
```

```
ABFTagGetTime session, 0, tag_time
```

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. ↗ *ABFTagGetCount, p. 94.*

The tag time is measured in seconds from the beginning of the experiment.

3.5.65 ABFTagGetType

ABFTagGetType returns the type of a tag.

Syntax

ABFTagGetType *session, tag, type*

session A Long value representing the current session.

tag A Long value representing the tag number to return.

type A Long variable in which the tag type is returned.

Example

```
Dim tag_type As Long
```

```
ABFTagGetType session, 0, tag_type
```

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. [↗ ABFTagGetCount, p. 94.](#)

The possible tag types are shown in table 59.

Table 59 Tag type values

Value	Type
0	Time
1	Comment
2	External
3	Voice

3.5.66 ABFTagGetVoiceNumber

ABFTagGetVoiceNumber returns the voice number of a tag.

Syntax

ABFTagGetVoiceNumber *session, tag, number*

session A Long value representing the current session.

tag A Long value representing the tag number to return.

number A Long variable in which the voice tag number is returned.

Example

```
Dim voice_number As Long
```

```
ABFTagGetType session, 0, voice_number
```

Discussion

You must have a file open.

Tags are numbered 0 to N-1, where N is the value returned by ABFTagGetCount. [↗ ABFTagGetCount, p. 94.](#)

The voice tag number is valid only if the associated tag is a voice tag. The tag type is returned by ABFTagGetType. [↗ ABFTagGetType, p. 95.](#)

4 Acquire: IGOR Pro

Section	
4.1 Script Interface	P. 96
4.2 Robust Processing	P. 100
4.3 Operations	P. 101
4.4 Reference	P. 102

4.1 Script Interface

This section explains how to access an Acquire data file. It contains a step by step description of the operations to perform to open a file and read the data in it.

4.1.1 Opening a File

Open an Acquire data file using the `AcquireFileOpen` operation. [☞ AcquireFileOpen, p. 111.](#)

In order to open a data file, you must supply the full path to the file. The IGOR PRO “Open” command can be used to bring up a file selection dialog and supply the resulting file path.

The following example brings up a file selection dialog to obtain a file specification. It then uses `AcquireFileOpen` to open the file for processing:

```
Variable fileReferenceNumber
Variable status

Open /D/R/T="BINFTEXT" fileReferenceNumber

if (strlen(S_fileName) > 0)
  AcquireFileOpen S_fileName, status
endif
```

4.1.2 File Parameters

Once you have opened a file, you can determine the parameters of the file. The parameters include the creation date and time, the file comment, and the sampling rate. [☞ File Parameter Operations, p. 101.](#)

For example, to display the file parameters in the history area:

```
Variable status
String fileDateTime
AcquireFileGetDateTime fileDateTime, status
Print fileDateTime
```

```
String fileComment
AcquireFileGetComment fileComment, status
Print fileComment
```

```
Variable sampleRate
AcquireFileGetSampleRate sampleRate, status
Print sampleRate
```

4.1.3 Series

A file is a sequence of series. You use the series operations to select a series and determine its parameters. [☞ Series Operations, p. 101.](#)

You use `AcquireSeriesGetCount` to determine the number of series in a file. [☞ AcquireSeriesGetCount, p. 119.](#)

Series are numbered 0 to N-1, where N is the number of series in a file. You use `AcquireSeriesSet` to set the series to use. [☞ AcquireSweepSet, p. 124.](#)

The following example determines the start of each series in a file. The start time is measured in seconds from the file

creation time:

```
Variable status
Variable series_count
Variable series
Variable start
```

```
AcquireSeriesGetCount series_count, status
series = 0
```

```
if (series_count > 0)
  do
    AcquireSeriesSet series, status
    AcquireSeriesGetStart start, status
    series += 1
    while (series < series_count)
  else
    Print "file contains no data"
  endif
```

A file may have no series if no data was acquired.

4.1.4 Sweeps

A series is a sequence of sweeps. You use the sweep operations to select a sweep and determine its parameters. [☞ Sweep Operations, p. 101.](#)

You use `AcquireSweepGetCount` to determine the number of sweeps in a file. [☞ AcquireSweepGetCount, p. 122.](#)

Sweeps are numbered 0 to N-1, where N is the number of sweeps in a file. You use `AcquireSweepSet` to set the sweep to use. [☞ AcquireSweepSet, p. 124.](#)

The following example prints the duration of each sweep in a series. The duration is measured in seconds:

```
Variable samplingRate
Variable sweepCount
Variable currentSweep
Variable sweepDuration
Variable status

AcquireFileGetSampleRate samplingRate
AcquireSeriesSet 0, status
AcquireSweepGetCount sweepCount, status
currentSweep = 0

do
  AcquireSweepSet currentSweep, status
```

```
AcquireSweepGetDuration sweepDuration, status
Print
  "Sweep ",
  currentSweep,
  " duration ",
  sweepDuration/samplingRate,
  "s"
  currentSweep += 1
while (currentSweep < sweepCount)
```

4.1.5 Segments

A sweep is a sequence of segments. If event screening was not used when a sweep was recorded, the sweep will consist of a single segment. You use the segment operations to select a segment and determine its parameters. [☞ Segment Operations, p. 102.](#)

You use `AcquireSegmentGetCount` to determine the number of segments in a sweep. The number will always be one if event screening was not used during sweep acquisition. [☞ AcquireSegmentGetCount, p. 116.](#)

Segments are numbered 0 to N-1, where N is the number of segments in a sweep. You use `AcquireSegmentSet` to set the segment number to use. [☞ AcquireSegmentSet, p. 118.](#)

The following example prints the starting sample number and number of samples in each segment of the current sweep.

```
Variable segmentCount
Variable currentSegment
Variable segmentStart
Variable segmentSamples
Variable status

AcquireSegmentGetCount segmentCount, status
currentSegment = 0

if (segmentCount > 0)
  do
    AcquireSegmentSet currentSegment, status
    AcquireSegmentGetStart segmentStart, status
    AcquireSegmentGetSampleCount
      segmentSamples, status
    Print
      "Segment ",
      currentSegment,
      "start",
      segmentStart,
```

```

    “ samples “,
    segmentSamples
    currentSegment += 1
    while (currentSegment < segmentCount)
    else
    Print “sweep contains no segments”
    endif

```

A sweep may have no segments if no data was acquired.

4.1.6 Channels

AcquireChannelGetCount returns the number of channels recorded in a file. [☞ AcquireChannelGetCount, p. 104.](#)

The following example prints the number of channels recorded in a file.

```

Variable channelCount
Variable status
AcquireChannelGetCount channelCount, status
Print channelCount

```

Channel numbers range from 0 to N-1, where N is the number of channels in the file.

4.1.7 Control Channels

Markers are associated with control channels. AcquireControlGetCount returns the number of control channels recorded in a file. [☞ AcquireControlGetCount, p. 107.](#)

To obtain information about a control channel use AcquireControlGet. [☞ AcquireControlGet, p. 106.](#)

The following example iterates through the control channels.

```

Variable type
String name
String signal
String description

```

```

String units
Variable control_count
Variable control
Variable status

AcquireControlGetCount control_count, status
control = 0

```

```

if (control_count > 0)
do
    AcquireControlGet control, type, name, description,
    units, signal,status

    | Process control channel here.
    control += 1
    while (control < control_count)
endif

```

4.1.8 Reading Data

The wave into which you read data must contain double-precision values.

Reading Segments

To read an entire segment, use AcquireSegmentRead. [☞ AcquireSegmentRead, p. 117.](#)

The following example reads each segment into a wave.

```

Variable segmentCount
Variable segmentSamples
Variable currentSegment
Variable status

AcquireSegmentGetCount segmentCount, status
currentSegment = 0

if (segmentCount > 0)
do
    AcquireSegmentSet currentSegment, status
    AcquireSegmentGetSampleCount
    segmentSamples, status
    Make /N=(segmentSamples)/D/O data
    AcquireSegmentRead channel, 0,
    segmentSamples, data, status
    | data processing goes here
    currentSegment += 1
    while (currentSegment < segmentCount)
endif

```

Reading Sections

If a segment is too large to read into memory at one time, you can read sections of a segment. To read a section, use `AcquireSegmentRead`. [☞ *AcquireSegmentRead*, p. 117.](#)

The following example reads each block of the current segment. It creates a single array, the size of the first block of the segment.

```
Variable blockCount
Variable currentBlock
Variable blockSamples
Variable startSample
Variable status

AcquireBlockGetSampleCount
  channel, 0, blockSamples, status
Make /N=(blockSamples)/D/O data

AcquireBlockGetCount blockCount, status
currentBlock = 0
startSample = 0

do
  AcquireBlockGetSampleCount
    channel, currentBlock, blockSamples, status
  AcquireSegmentRead
    channel, startSample, blockSamples, data, status
  | data processing goes here
  currentBlock += 1
  startSample += blockSamples
while (currentBlock < blockCount)
```

4.1.9 Markers

Markers are stored by sweep. To determine the number of markers in the current sweep, use `AcquireMarkerGetCount`. [☞ *AcquireMarkerGetCount*, p. 114.](#)

Each marker has optional associated text and data. To obtain information about a marker use `AcquireMarkerGet`. To obtain the marker text use `AcquireMarkerGetText`. [☞ *AcquireMarkerGet*, p. 113.](#) [☞ *AcquireMarkerGetText*, p. 115.](#)

The following example iterates through the markers in a

sweep:

```
Variable control_channel
String text
Variable sample
Variable value
Variable data_size
Variable extended_data_size
Variable marker
Variable status

AcquireMarkerGetCount marker_count, status
marker = 0

if (marker_count > 0)
  do
    AcquireMarkerGet marker, control_channel,
    sample,
    value, data_size, extended_data_size, status

    AcquireMarkerGetText marker, text, status

    // Process marker here.
    marker += 1
  while (marker < marker_count)
endif
```

4.1.10 Closing a File

To close a file, use `AcquireFileClose`. [☞ *AcquireFileClose*, p. 108.](#)

The following example closes an Acquire file.

```
AcquireFileClose
```

4.1.11 Error Handling

The examples presented in this section take no account of errors that may occur, such as when `AcquireFileOpen` is called for a file that does not contain Acquire data.

If you write IGOR Pro procedures to be used by someone else, you will probably be interested in how to handle error conditions in your procedures so you can provide better error recovery for the user. This manual contains information about how to make your procedures more robust. [☞ *Robust Processing*, p. 100.](#)

4.1.12 Further Information

This manual contains a complete description of each DataAccess Pro operation. These descriptions are useful to answer specific questions regarding the individual operations. *☞ Reference, p. 102.*

4.2 Robust Processing

If you write IGOR Pro procedures to be used by others, you may find it helpful to handle error conditions within your procedures, recovering in a manner that allows the user to proceed with their operations.

This section explains how to use the information provided by DataAccess Pro to recover from error conditions in your IGOR Pro procedures.

4.2.1 Errors

The status information provided by DataAccess Pro allows you to recover from error conditions caused by the program, user or the system. It does not allow you to recover from syntax errors in your program.

Program Errors

Errors in your program include such mistakes as using `AcquireSegmentSet` to set an invalid segment number. You can determine the range of valid segment numbers using `AcquireSegmentGetCount`, so you can avoid setting an invalid segment number. Catching and reporting such errors is helpful in debugging a procedure.

System Errors

System errors include such problems as an error reading an Acquire data file. Such errors are usually the result of problems such as a corrupted disk, a missing floppy or CD-ROM, or defective hardware such as a disk drive. Catching such errors prevents further problems and may allow the procedure to recover, for example by allowing the user to insert a missing file medium.

User Errors

User errors include mistakes such as when the user selects a

file that is not an Acquire data file. The recovery allows your procedures to provide an error indication to the user and allow the user to select a different file.

Syntax Errors

Syntax errors include mistakes such as passing the wrong number or types of parameters to an operation. An improperly dimensioned wave is also treated as a syntax error. Such errors are returned directly to Igor Pro and cause a procedure to terminate.

4.2.2 Status

Nearly all operations return a status value through a status parameter. The status value is non-zero if an error occurred. All DataAccess Pro and system errors are returned in this manner. Such errors will not terminate a procedure. The procedure should be designed to handle such conditions. Igor Pro errors, such as a missing parameter, are returned directly to Igor and will terminate a procedure.

Status values for specific errors are not defined, and may change from version to version of DataAccess Pro. The `AcquireGetStatusText` operation translates a status value to text. *☞ AcquireGetStatusText, p. 112.*

The following example allows the user to select a file, and does not terminate until the user selects a valid Acquire file.

```

Variable fileNumber
Variable status
String message

do
  Open /D/R/T="BINFTEXT" fileNumber

  if (strlen(S_fileName) > 0)
    AcquireFileOpen S_fileName, status

    if (status != 0)
      AcquireGetStatusText status, message
      Print "AcquireFileOpen error: ", message
    endif
  endif
while (status != 0)

```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

Only selected operations provide a status value.

4.3 Operations

This chapter describes each of the operations provided by DataAccess Pro. The operations are grouped by category.

4.3.1 Common Operations

The common operations are always valid, regardless of whether an Acquire file is open or not. The common operations are shown in table 60.

Table 60 Common operations

Operation	
AcquireGetVersion	p. 112
AcquireGetStatusText	p. 112

4.3.2 File Operations

The file operations allow you to access an Acquire data file. The file operations are shown in table 61.

Table 61 File operations

Operation	
AcquireFileOpen	p. 111
AcquireFileClose	p. 108

4.3.3 File Parameter Operations

The file parameter operations allow you to obtain file parameters. The file parameter operations are shown in table 62.

Table 62 File parameter operations

Operation	
AcquireFileGetComment	p. 109
AcquireFileGetSampleRate	p. 109
AcquireFileGetTechnique	p. 110
AcquireFileGetTime	p. 110
AcquireFileGetVersion	p. 111

The file parameter operations are valid when a file is open. In this case the returned parameters are the parameters of

the open file.

4.3.4 Channel Operations

The channel operations allow you to determine the number of channels in a file, and obtain channel parameters. The channel operations are shown in table 63.

Table 63 Channel operations

Operation	
AcquireChannelGetCount	p. 104
AcquireChannelGetDescription	p. 104
AcquireChannelGetRange	p. 105
AcquireChannelGetScaling	p. 105

4.3.5 Series Operations

The series operations allow you to access each series of a file. The series operations are shown in table 64.

Table 64 Series operations

Operation	
AcquireSeriesGetCount	p. 119
AcquireSeriesSet	p. 121
AcquireSeriesGetComment	p. 119
AcquireSeriesGetStart	p. 120
AcquireSeriesGetTechnique	p. 120

4.3.6 Sweep Operations

The sweep operations allow you to access each sweep of a file. The sweep operations are shown in table 65.

Table 65 Sweep operations

Operation	
AcquireSweepGetCount	p. 122
AcquireSweepSet	p. 124
AcquireSweepGetComment	p. 121
AcquireSweepGetDuration	p. 122
AcquireSweepGetScreen	p. 123
AcquireSweepGetStart	p. 123
AcquireSweepGetTechnique	p. 124

4.3.7 Segment Operations

The segment operations allow you to access each segment of a sweep. If a file was acquired without event screening, each sweep will have a single segment.

The segment operations are shown in table 66.

Table 66 Segment operations

Operation	
AcquireSegmentGetCount	p. 116
AcquireSegmentSet	p. 118
AcquireSegmentGetSampleCount	p. 116
AcquireSegmentGetStart	p. 117
AcquireSegmentRead	p. 117

4.3.8 Block Operations

The block operations allow you to determine the parameters of a block. You do not normally need these operations.

The block operations are shown in table 67.

Table 67 Block operations

Operation	
AcquireBlockGetCount	p. 103
AcquireBlockGetSampleCount	p. 103

4.3.9 Control Channel Operations

The control channel operations allow you to determine the number of control channels and to obtain the control parameters. The control channel operations are shown in table 68.

Table 68 Control operations

Operation	
AcquireControlGet	p. 106
AcquireControlGetCount	p. 107
AcquireControlGetHoldingValue	p. 108

4.3.10 Marker Operations

The marker operations allow you to determine the number of markers in the current sweep and to obtain the marker parameters and data. The marker operations are shown in

table 69.

Table 69 Marker operations

Operation	
AcquireMarkerGet	p. 113
AcquireMarkerGetCount	p. 114
AcquireMarkerGetData	p. 114
AcquireMarkerGetExtendedData	p. 115
AcquireMarkerGetText	p. 115

4.4 Reference

This section lists each DataAccess Pro operation in alphabetical order.

4.4.1 AcquireBlockGetCount

AcquireBlockGetCount returns the number of blocks in the current segment.

Syntax

AcquireBlockGetCount *channel, count, status*

channel The channel to use.

count The name of the variable in which to return the number of blocks in the current segment.

status A variable to receive the status of the operation.

Example

Variable status
Variable blockCount

AcquireSweepSet 8
AcquireSegmentSet 3

AcquireBlockGetCount channel, blockCount, status

Discussion

A file must be open and a current segment selected. To set the current segment, use AcquireSegmentSet.
☞ *AcquireSegmentSet, p. 118.*

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount.
☞ *AcquireChannelGetCount, p. 104.*

4.4.2 AcquireBlockGetSampleCount

AcquireBlockGetSampleCount returns the number of samples in the specified block.

Syntax

AcquireBlockGetSampleCount *channel, block, samples, status*

channel The channel to use.

block The block number to use.

samples The name of the variable in which to return the number of samples in the current block.

status A variable to receive the status of the operation.

Example

Variable status
Variable blockSamples

AcquireSweepSet 0
AcquireSegmentSet 0

AcquireBlockGetSampleCount 0, 0, blockSamples,
status

Discussion

You must have a file open, and a current segment selected. To set the current segment, use AcquireSegmentSet.
☞ *AcquireSegmentSet, p. 118.*

For a given channel, all blocks in a file have the same number of samples, except the last block of each segment, which may have fewer samples.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount.
☞ *AcquireChannelGetCount, p. 104.*

Blocks are numbered 0 to N-1, where N is the number of blocks in the segment. To determine the number of blocks in the current segment, use AcquireBlockGetCount.
☞ *AcquireBlockGetCount, p. 103.*

4.4.3 AcquireChannelGetCount

AcquireChannelGetCount returns the number of channels recorded in the current Acquire data file.

Syntax

AcquireChannelGetCount *count, status*

count The name of the variable in which to return the number of channels.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable channelCount
AcquireChannelGetCount channelCount, status
```

Discussion

The channel count is the number of active channels recorded in the Acquire data file. For example, if data was recorded from 16 channels 0, 2, and 5, the channel count is 3. The channel count must be at least 1.

4.4.4 AcquireChannelGetDescription

AcquireChannelGetDescription returns a description of the specified channel.

Syntax

AcquireChannelGetDescription *channel, label, signal, status*

channel A value representing the channel to use.

label The name of the string variable in which the channel label is returned.

signal The name of the string variable in which the channel signal is returned.

status A variable to receive the status of the operation.

Example

```
String label
String signal

AcquireChannelGetDescription 0, label, signal, status
```

 *Channel Operations, p. 101.*

Discussion

You must have a file open.

The channel label is supplied by Acquire, and is the same as the front-panel label of the channel on the data acquisition device.

The channel signal is supplied by the user.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount.

 *AcquireChannelGetCount, p. 104.*

4.4.5 AcquireChannelGetRange

AcquireChannelGetRange returns the range of the specified channel.

Syntax

AcquireChannelGetRange *channel, minimum, maximum, status*

channel The channel to use.

minimum The name of the variable to receive the lower limit.

maximum The name of the variable to receive the upper limit.

status A variable to receive the status of the operation.

Example

Variable status
Variable minimum
Variable maximum

AcquireChannelGetRange 0, minimum, maximum, status

Discussion

You must have a file open.

AcquireChannelGetRange returns the minimum and maximum possible values for the specified channel. The range of the acquisition device and the user supplied gain are taken into account. [☞ AcquireChannelGetScaling, p. 105.](#)

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount. [☞ AcquireChannelGetCount, p. 104.](#)

4.4.6 AcquireChannelGetScaling

AcquireChannelGetScaling returns the scaling for the specified channel.

Syntax

AcquireChannelGetScaling *channel, gain, units, natural_scale, status*

channel A value representing the channel to use.

gain name of the variable to receive the gain.

units The name of the string variable in which the channel units are returned.

natural_scale The name of the variable to receive the natural scale.

Example

Variable gain
Variable scale
String units

AcquireChannelGetScaling 0, gain, units, scale, status

[☞ Channel Operations, p. 101.](#)

Discussion

You must have a file open.

The channel gain is supplied by Acquire from a value entered by the user.

The channel units are supplied by the user.

The natural scale is a power of 1000 which indicates the natural scale of the measured signal. For example, if the natural scale is pA then AcquireChannelGetNaturalScale will return -4, since $1000^{-4} = 10^{-12}$. The natural scale then corresponds to the standard prefix. Common scalings

are shown in table 70.

Table 70 Common natural scale values

Natural scale	Scale	Prefix
-4	10^{-12}	p (pico)
-3	10^{-9}	n (nano)
-2	10^{-6}	u (micro)
-1	10^{-3}	m (milli)
0	1	
1	10^3	k (kilo)
2	10^6	M (mega)
3	10^9	G (giga)

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use `AcquireChannelGetCount`.
 ☞ *AcquireChannelGetCount*, p. 104.

4.4.7 AcquireControlGet

`AcquireControlGet` returns the parameters for the specified control.

Syntax

`AcquireControlGet control, name, description, units, natural_scale, signal, status`

control A value representing the control index, that is, the number of the control.

name A variable to receive the control name.

description A String variable to receive the control description.

units A String variable to receive the control value units.

natural_scale The name of the variable to receive the natural scale.

signal A String variable to receive the control value name.

status A variable to receive the status of the operation.

Example

```
String name
String description
String units
Variable scale
String signal
```

```
AcquireControlGet 0, name, description, units, scale,
signal, status
```

Discussion

You must have a file open.

Controls are numbered 0 to N-1, where N is the number of controls used with the file. To determine the number of controls in the file, use `AcquireControlGetCount`.
 ☞ *AcquireControlGetCount*, p. 107.

The natural scale is a power of 1000 which indicates the natural scale of the control signal. For example, if the natural scale is pA (10^{-12} A), then the `natural_scale` value returned will be -4, since $1000^{-4} = 10^{-12}$. The natural

scale value corresponds to the standard prefix. Common scalings are shown in table 71. These are not the only scalings that may be supported by Acquire.

Table 71 Common natural scale values

Natural scale	Scale	Prefix
-4	10^{-12}	p (pico)
-3	10^{-9}	n (nano)
-2	10^{-6}	u (micro)
-1	10^{-3}	m (milli)
0	1	
1	10^3	k (kilo)
2	10^6	M (mega)
3	10^9	G (giga)

4.4.8 AcquireControlGetCount

AcquireControlGetCount returns the number of controls used in the current Acquire data file.

Syntax

AcquireControlGetCount *count, status*

count A variable to receive the number of controls.

status A variable to receive the status of the operation.

Example

Variable count

AcquireControlGetCount count, status

Discussion

You must have a file open.

The control count is the number of active controls used with the Acquire data file.

4.4.9 AcquireControlGetHoldingValue

AcquireControlGetHoldingValue returns the holding value for the specified control for the current sweep.

Syntax

AcquireControlGetHoldingValue *control, value, status*

control A value representing the control index.

value A variable to receive the holding value.

status A variable to receive the status of the operation.

Example

Variable value

AcquireControlGetHoldingValue 0, value, status

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

At the end of a sweep, a control output returns to a holding value. The holding value has an associated value to record in the file. This function returns that associated value.

The holding value may vary between sweeps.

Controls are numbered 0 to N-1, where N is the number of controls used with the file. To determine the number of controls in the file, use AcquireControlGetCount. [☞ AcquireControlGetCount, p. 107.](#)

4.4.10 AcquireFileClose

AcquireFileClose terminates processing of an Acquire file.

Syntax

AcquireFileClose

Example

AcquireFileClose

Discussion

You should perform AcquireFileClose as soon as you complete processing of an Acquire data file. The operation releases the memory required for the Acquire file directory.

4.4.11 AcquireFileGetComment

AcquireFileGetComment returns the comment for the file.

Syntax

AcquireFileGetComment *comment, status*

comment The name of the string variable in which the file comment is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
String fileComment
AcquireFileGetComment fileComment, status
```

Discussion

You must have a file open.

The user enters the file comment in Acquire.

4.4.12 AcquireFileGetSampleRate

AcquireFileGetSampleRate returns the number of samples taken per second on each channel.

Syntax

AcquireFileGetSampleRate *rate, status*

rate The name of the variable in which the sample rate is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable sampleRate
AcquireFileGetSampleRate sampleRate, status
```

Discussion

You must have a file open.

All channels are sampled at the same rate, and the sampling rate is the same throughout the file.

The value returned is the number of samples taken per second on each channel. If three channels are sampled, the total sampling rate is the sampling rate on each channel times three.

4.4.13 AcquireFileGetTechnique

AcquireFileGetTechnique returns the technique for the file.

Syntax

AcquireFileGetTechnique *technique, status*

technique The name of the string variable in which the file technique string is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
String technique
AcquireFileGetTechnique technique, status
```

Discussion

The technique represents information provided during recording that is intended to be used during data analysis. The technique string can only be set through Acquire scripting. It cannot be set through the Acquire user interface.

You must have a file open.

The user enters the file technique string in Acquire.

4.4.14 AcquireFileGetTime

AcquireFileGetTime returns the creation date and time of the file as a string.

Syntax

AcquireFileGetTime *time, status*

time The name of the string variable in which the file creation date and time are returned. The format of the string depends on your system settings.

status A variable to receive the status of the operation.

Example

```
Variable status
String fileDateTime
AcquireFileGetTime fileDateTime, status
```

Discussion

You must have a file open.

4.4.15 AcquireFileGetVersion

AcquireFileGetVersion returns the Acquire format version of the file.

Syntax

AcquireFileGetVersion *version, status*

version A String variable in which the file version is returned.

status A variable to receive the status value.

Example

Variable status
String version

```
AcquireFileGetVersion version, status
```

Discussion

You must have a file open.

4.4.16 AcquireFileOpen

AcquireFileOpen prepares an Acquire data file for processing.

Syntax

AcquireFileOpen *path, status*

path The full path of the file.

status A variable to receive the status of the operation.

Example

The following sequence brings up a file selection dialog using the IGOR “open” command. It passes the resulting path to AcquireFileOpen.

Variable fileReferenceNumber
Variable status

```
Open /D/R/T="BINFTEXT" fileReferenceNumber
```

```
if (strlen(S_fileName) > 0)
    AcquireFileOpen S_fileName, status
endif
```

Discussion

AcquireFileOpen loads the Acquire file directory into memory. This directory describes the location of each sweep and segment in the file, and makes access to any part of the file efficient. The directory can require a substantial amount of memory, so it is possible that AcquireFileOpen will fail due to limited memory.

The status is zero if the operation succeeded, non-zero if it failed. Use AcquireGetStatusText to interpret the status. [☞ AcquireGetStatusText, p. 112.](#)

Once a file is open, you can access file parameters. [☞ File Parameter Operations, p. 101.](#) You can process the sweeps in the file to access the data. [☞ Sweep Operations, p. 101.](#)

When you are done with a file, you should close it. [☞ AcquireFileClose, p. 108.](#)

AcquireFileOpen invalidates the current sweep. You must set the current sweep using AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

4.4.17 AcquireGetStatusText

AcquireGetStatusText converts a status value to a text string.

Syntax

AcquireGetStatusText *status, message*

status A status value returned by DataAccess Pro.

message A string variable to receive the text corresponding to the status value.

Example

The following sequence attempts to open a file and prints a message if an error occurs:

```
String message
Variable status

AcquireFileOpen "Test.dat", status

if (status != 0)
    AcquireGetStatusText status, message
    Print "AcquireFileOpen error:", message
endif
```

Discussion

AcquireGetStatusText is the only means provided to interpret status values.  *Robust Processing*, p. 100.

4.4.18 AcquireGetVersion

AcquireGetVersion returns the version number of the DataAccess Pro package.

Syntax

AcquireGetVersion *version*

version The name of the variable to receive the version number.

Example

```
Variable version
AcquireGetVersion version
```

Discussion

If you are writing a set of procedures using DataAccess Pro, you can use AcquireGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess Pro. You can use the following code to ensure that this version or a later version is in use:

```
Variable version
AcquireGetVersion version

if (version < N)
    print "incorrect version of DataAccess Pro"
endif
```

4.4.19 AcquireMarkerGet

AcquireMarkerGet returns the parameters for the specified marker.

Syntax

AcquireMarkerGet *marker, control, sample, value, data_size, extended_data_size, status*

marker An Long value representing the marker for which to return the parameters.

control A Long variable to receive the index of the control which created the marker.

sample A Double variable to receive the sample number from the beginning of the sweep indicating the time of the marker.

value A Double variable to receive the marker value.

data_size A Long variable to receive the size in bytes of the data associated with the marker.

extended_data_size A Long variable to receive the size in bytes of the extended data associated with the marker.

status A variable to receive the status of the operation.

Example

Variable status
 Variable control
 Variable value
 Variable sample
 Variable data_size
 Variable extended_data_size

AcquireMarkerGet marker, control, sample,
 value, data_size, extended_data_size, status

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. *☞ AcquireSweepSet, p. 124.*

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use AcquireMarkerGetCount.

☞ AcquireMarkerGetCount, p. 114.

The data will be at most 256 bytes in length.

The extended data is limited in size only by the memory storage available on your system.

4.4.20 AcquireMarkerGetCount

AcquireMarkerGetCount returns the number of markers contained in the current sweep.

Syntax

AcquireMarkerGetCount *count, status*

count A variable to receive the number of markers in the current sweep.

status A variable to receive the status of the operation.

Example

Variable status

Variable count

AcquireMarkerGetCount count, status

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

4.4.21 AcquireMarkerGetData

AcquireMarkerGetData returns the data associated with the marker.

Syntax

AcquireMarkerGetData *marker, data, status*

marker A value representing the marker from which to get the data.

data A wave to receive the data. The wave can be of any data type except for text. The wave must be large enough to store the data.

status A variable to receive the status of the operation.

Example

Variable status

Make/N=32/D data

AcquireMarkerGetData 0, data, status

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use AcquireMarkerGetCount. [☞ AcquireMarkerGetCount, p. 114.](#)

Use AcquireMarkerGet to determine the size of the data stored with the marker. [☞ AcquireMarkerGet, p. 113.](#)

4.4.22 AcquireMarkerGetExtendedData

AcquireMarkerGetExtendedData returns the extended data associated with the marker.

Syntax

AcquireMarkerGetExtendedData *marker, data, status*

marker A value representing the marker from which to get the data.

data A wave to receive the extended data. The wave can be of any data type except for text. The wave must be large enough to store the extended data.

status A variable to receive the status of the operation.

Example

Variable status
Variable control
Variable value
Variable sample
Variable data_size
Variable extended_data_size

AcquireMarkerGet marker, control, sample,
value, data_size, extended_data_size, status

Make/N=(extended_data_size/2)/W data

AcquireMarkerGetExtendedData 0, data, status

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use AcquireMarkerGetCount. [☞ AcquireMarkerGetCount, p. 114.](#)

Use AcquireMarkerGet to determine the size of the extended data stored with the marker. The extended data is limited in size only by the memory storage available on your system. [☞ AcquireMarkerGet, p. 113.](#)

4.4.23 AcquireMarkerGetText

AcquireMarkerGetText returns the text associated with the marker.

Syntax

AcquireMarkerGetText *session, marker, text, status*

marker A value representing the marker from which to get the data.

text A String variable to receive the text.

status A variable to receive the status of the operation.

Example

Variable status
String marker_text

AcquireMarkerGetText session, 0, marker_text, status

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use AcquireMarkerGetCount. [☞ AcquireMarkerGetCount, p. 114.](#)

Use AcquireMarkerGet to determine the size of the text stored with the marker. The text will be at most 256 characters in size. [☞ AcquireMarkerGet, p. 113.](#)

4.4.24 AcquireSegmentGetCount

AcquireSegmentGetCount returns the number of segments in the current sweep.

Syntax

AcquireSegmentGetCount *count, status*

count The name of the variable in which the segment count is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable segmentCount
AcquireSweepSet 5
AcquireSegmentGetCount segmentCount, status
```

Discussion

You must have a file open and a current sweep selected. You can set the current sweep using AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

You can set the current segment using AcquireSegmentSet. [☞ AcquireSegmentSet, p. 118.](#)

The segment count is the same for all channels in the sweep.

4.4.25 AcquireSegmentGetSampleCount

AcquireSegmentGetSampleCount returns the number of samples in the current segment.

Syntax

AcquireSegmentGetSampleCount *samples, status*

samples The name of the variable in which the segment size is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable samples
AcquireSegmentSet 4
AcquireSegmentGetSampleCount samples, status
```

Discussion

You must have a file open and a segment selected. You can select the current segment using AcquireSegmentSet. [☞ AcquireSegmentSet, p. 118.](#)

The number of samples in a segment is the same for all channels.

4.4.26 AcquireSegmentGetStart

AcquireSegmentGetStart returns the time of the current segment within the current sweep, measured in samples.

Syntax

AcquireSegmentGetStart *start, status*

start The name of the variable in which the starting sample number is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable segmentStart
AcquireSegmentGetStart segmentStart, status
```

Discussion

You must have a file open and a segment selected. You can select the current segment using AcquireSegmentSet. [☞ AcquireSegmentSet, p. 118.](#)

A segment is contained completely within a sweep. The starting sample number of a segment plus the number of samples in a segment will be less than or equal to the duration of the containing sweep, measured in samples. AcquireSegmentGetSampleCount returns the number of samples in the current segment. [☞ AcquireSegmentGetSampleCount, p. 116.](#) AcquireSweepGetDuration returns the duration of the current sweep. [☞ AcquireSweepGetDuration, p. 122.](#)

4.4.27 AcquireSegmentRead

AcquireSegmentRead reads the specified section of a segment into a wave.

Syntax

AcquireSegmentRead *channel, sample, count, data, status*

channel The channel to use.

sample The starting sample number within the segment.

count The number of samples to read.

data The name of an existing wave to receive the data. The wave must contain double-precision real values.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable samples
Variable start
Variable channel
```

| *Set the start, samples, and channel here*

```
Make /n=(samples) /d channel1Data
AcquireSegmentRead
channel, start, samples, channel1Data, status
```

Discussion

A file must be open, and a sweep and segment selected.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount. [☞ AcquireChannelGetCount, p. 104.](#)

The wave must be long enough to contain the specified number of samples. You can create a wave using the IGOR “make” operation.

Reading a section is more efficient if it contains exactly one or more blocks. [☞ Block Operations, p. 102.](#)

When AcquireSegmentReadSection reads the data, it sets

the following properties of the wave:

- 1 The y axis units are set to 'V' for volts.
- 2 The x axis units are set to 's' for seconds.
- 3 The x axis increment is set to the sampling interval.

With these settings, the wave data will appear properly if graphed.

4.4.28 AcquireSegmentSet

AcquireSegmentSet sets the segment number to use within the current sweep.

Syntax

AcquireSegmentSet *segment*, *status*

segment The number of the segment to use.

status A variable to receive the status of the operation.

Example

```
Variable status  
AcquireSegmentSet 0, status
```

Discussion

You must have a file open and a sweep selected.
☞ *AcquireSweepSet*, p. 124.

If a sweep contains N segments, segment numbers for the sweep are in the range 0 to N-1. You can obtain the number of segments in the current sweep using AcquireSegmentGetCount. ☞ *AcquireSegmentGetCount*, p. 116.

If a file was recorded without event screening, each sweep contains only one segment, and the only valid segment number is 0.

4.4.29 AcquireSeriesGetComment

AcquireSeriesGetComment returns the comment for the current series.

Syntax

AcquireSeriesGetComment *comment, status*

comment The name of the string variable in which the series comment is returned.

status A variable to receive the status of the operation.

Example

Variable status

String text

AcquireSeriesGetComment text, status

Discussion

You must have a file open and a series selected.

To set the current series, use AcquireSeriesSet.
 ☞ *AcquireSeriesSet, p. 121.*

4.4.30 AcquireSeriesGetCount

AcquireSeriesGetCount returns the number of series in the open file.

Syntax

AcquireSeriesGetCount *count, status*

count The name of the variable in which the count is returned.

status A variable to receive the status of the operation.

Example

Variable status

Variable series_count

AcquireSeriesGetCount series_count, status

```
if (series_count = 0)
    // Handle an empty file
endif
```

Discussion

You must have a file open.

The series count may be zero. This indicates that no data was recorded in the file.

To set the current series, use AcquireSeriesSet.
 ☞ *AcquireSeriesSet, p. 121.*

4.4.31 AcquireSeriesGetStart

AcquireSeriesGetStart returns the time at which the current series starts within the file, measured in seconds.

Syntax

AcquireSeriesGetStart *start, status*

start The name of the variable in which the starting time is returned.

status A variable to receive the status of the operation.

Example

Variable status
Variable start

AcquireSeriesGetStart start, status

Discussion

The start time of a series is measured in seconds since the file was created, not necessarily since the start of data acquisition. The time is not measured to an accuracy greater than one second.

4.4.32 AcquireSeriesGetTechnique

AcquireSeriesGetTechnique returns the technique for the current series.

Syntax

AcquireSeriesGetTechnique *technique, status*

technique The name of the string variable in which the technique is returned.

status A variable to receive the status of the operation.

Example

Variable status
String text

AcquireSeriesGetTechnique text, status

Discussion

The technique represents information provided during recording that is intended to be used during data analysis. The technique string can only be set through Acquire scripting. It cannot be set through the Acquire user interface.

You must have a file open and a series selected.

To set the current series, use AcquireSeriesSet.
☞ *AcquireSeriesSet*, p. 121.

4.4.33 AcquireSeriesSet

AcquireSeriesSet sets the series number to use.

Syntax

AcquireSeriesSet *series, status*

series The series number to use.

status A variable to receive the status of the operation.

Example

Variable status
AcquireSeriesSet 0, status

Discussion

You must have a file open.

If a file contains N series, series numbers are in the range 0 to N-1. You can obtain the number of series in the current file using AcquireSeriesGetCount. [☞ AcquireSeriesGetCount, p. 119.](#)

AcquireSeriesSet invalidates the current sweep. You must set the current sweep using AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

4.4.34 AcquireSweepGetComment

AcquireSweepGetComment returns the comment for the current sweep.

Syntax

AcquireSweepGetComment *comment, status*

comment The name of the string variable in which the file comment is returned.

status A variable to receive the status of the operation.

Example

Variable status
Variable text

AcquireSweepGetComment text, status

Discussion

You must have a file open and a series and sweep selected.

To set the current series, use AcquireSeriesSet. [☞ AcquireSeriesSet, p. 121.](#)

To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

4.4.35 AcquireSweepGetCount

AcquireSweepGetCount returns the number of sweeps in the open file.

Syntax

AcquireSweepGetCount *count, status*

count The name of the variable in which the sweep count is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable sweepCount
AcquireSweepGetCount sweepCount, status
```

Discussion

You must have a file open.

The sweep count may be zero. This indicates that no data was recorded in the file.

To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

4.4.36 AcquireSweepGetDuration

AcquireSweepGetDuration returns the duration of the current sweep, measured in samples.

Syntax

AcquireSweepGetDuration *duration, status*

duration The name of the variable in which the sweep duration is returned.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable sweepDuration
AcquireSweepSet 0
AcquireSweepGetDuration sweepDuration, status
```

Discussion

You must have a file open and a current sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 124.](#)

The duration of a sweep is measured in samples, and time within a sweep is measured in samples. To obtain the starting sample of a segment within a sweep, use AcquireSegmentGetStart. [☞ AcquireSegmentGetStart, p. 117.](#)

If event screening was not used during data acquisition, the sweep will contain only one segment, and the size of that segment will be the same as the duration of the sweep.

If event screening was used during data acquisition, the sweep may contain discontinuous segments. In that case the sweep duration will be greater than the sum of the segment sizes.

4.4.37 AcquireSweepGetScreen

AcquireSweepGetScreen returns a description of the screening used while recording the current sweep.

Syntax

AcquireSweepGetScreen *screen, status*

screen The name of the string variable in which the sweep screening description is returned.

status A variable to receive the status of the operation.

Example

Variable status
Variable screen

AcquireSweepGetScreen screen, status

Discussion

You must have a file open and a series and sweep selected.

To set the current series, use AcquireSeriesSet.
☞ *AcquireSeriesSet*, p. 121.

To set the current sweep, use AcquireSweepSet.
☞ *AcquireSweepSet*, p. 124.

4.4.38 AcquireSweepGetStart

AcquireSweepGetStart returns the time at which the current sweep starts within the current series, measured in seconds.

Syntax

AcquireSweepGetStart *start, status*

start The name of the variable in which the starting time is returned.

status A variable to receive the status of the operation.

Example

Variable status
Variable startTime
AcquireSweepGetStart startTime, status

Discussion

The start time of a sweep is measured in seconds since the file was created, not necessarily since the start of data acquisition. The time is not measured to an accuracy greater than one second.

4.4.39 AcquireSweepGetTechnique

AcquireSweepGetTechnique returns the technique used to record the current sweep.

Syntax

AcquireSweepGetTechnique *technique, status*

technique The name of the string variable in which the sweep technique is returned.

status A variable to receive the status of the operation.

Example

Variable status
Variable technique

AcquireSweepGetTechnique technique, status

Discussion

The technique represents information provided during recording that is intended to be used during data analysis. The technique string can only be set through Acquire scripting. It cannot be set through the Acquire user interface.

You must have a file open and a series and sweep selected. To set the current series, use AcquireSeriesSet. To set the current sweep, use AcquireSweepSet. [☞ AcquireSeriesSet, p. 121.](#) [☞ AcquireSweepSet, p. 124.](#)

4.4.40 AcquireSweepSet

AcquireSweepSet sets the sweep number to use.

Syntax

AcquireSweepSet *sweep, status*

sweep The number of the sweep to use.

status A variable to receive the status of the operation.

Example

Variable status
AcquireSweepSet 0, status

Discussion

You must have a file open.

If a file contains N sweeps, sweep numbers are in the range 0 to N-1. You can obtain the number of sweeps in the current file using AcquireSweepGetCount. [☞ AcquireSweepGetCount, p. 122.](#)

AcquireSweepSet invalidates the current segment. You must set the current segment using AcquireSegmentSet. [☞ AcquireSegmentSet, p. 118.](#)

5 Acquire: Visual Basic

Section	
5.1 Using DataAccess	P. 125
5.2 Getting Started	P. 125
5.3 Robust Processing	P. 129
5.4 Operations	P. 130
5.5 Reference	P. 131

5.1 Using DataAccess

To make DataAccess available to your program, add the file `AcquireVB.bas` to your project. Ensure that the file `AcquireVB.dll` is in your path.

To make DataAccess available to an Systat Software SigmaPlot script module, include the following statement in the module:

```
'#Uses "AcquireVB.bas"
```

The leading quote (!) is required. The bas file must be in the same folder as your SigmaPlot project, or you must explicitly specify the path to the bas file.

5.1.1 Status

Each operation returns a status code. This code is zero if no error is detected.

If an error is detected, the returned status value is non-zero. It can be translated to a text string using `AcquireGetStatusText`. [☞ *AcquireGetStatusText*, p. 142.](#)

5.1.2 Data Types

Integer values are passed as the data type “Long”. Real values are passed as the data type “Double”. Arrays are passed by passing the first element in the array. Character string values are passed as the data type “Variant”. The data type “String” is not used because when Visual Basic passes parameters to dlls, it performs undesired translations from Unicode to ASCII.

5.1.3 Calls

Almost all DataAccess procedures return a value. In Visual Basic, you have several choices of syntax when you call such procedures. For example, you can call `AcquireChannelGetCount` as follows:

- 1 As a function, using the return value:

```
result = AcquireChannelGetCount(session, count)
```

- 2 As a subroutine using “Call”:

```
Call AcquireChannelGetCount(session, count)
```

- 3 Directly as a command. In this case, the parameters are not enclosed in parentheses:

```
AcquireChannelGetCount session, count
```

Any of the methods work. The examples in this manual generally use the command form.

5.2 Getting Started

This section explains how to access an Acquire data file. It contains a step by step description of the operations to perform to open a file and read the data in it.

5.2.1 Opening a Session

Open a session using the `AcquireSessionOpen` operation. [☞ *AcquireSessionOpen*, p. 151.](#)

The AcquireSessionOpen operation returns a handle. You must pass this handle to other operations that use that session.

5.2.2 Opening a File

Open an Acquire data file using the AcquireFileOpen operation. [☞ AcquireFileOpen, p. 141.](#)

In order to open a data file, you must supply a file path.

The following example opens a session, then opens a file with the path “f:\test.dat”. After the file is open, it closes the file and the session.

```
Dim session As Long
Dim status As Long
Dim file_name As Variant
file_name = "f:\test.dat"

AcquireSessionOpen session
status = AcquireFileOpen(session, file_name)

If status <> 0 Then
    ' The file could not be opened. Handle the
    ' error and do not call AcquireFileClose
End If

AcquireFileClose session
AcquireSessionClose session
```

5.2.3 File Parameters

Once you have opened a file, you can determine the parameters of the file. The parameters include the creation date and time, the file comment, and the sampling rate. [☞ File Parameter Operations, p. 130.](#)

The following example obtains a set of file parameters:

```
Dim file_date_time As Date
Dim sample_rate As Double

AcquireFileGetDateTime session, file_date_time
AcquireFileGetSampleRate session, sample_rate
```

5.2.4 Series

A file is a sequence of series. You use the series operations

to select a series and determine its parameters. [☞ Series Operations, p. 130.](#)

You use AcquireSeriesGetCount to determine the number of series in a file. [☞ AcquireSeriesGetCount, p. 149.](#)

Series are numbered 0 to N-1, where N is the number of series in a file. You use AcquireSeriesSet to set the series to use. [☞ AcquireSweepSet, p. 155.](#)

The following example determines the start of each series in a file. The start time is measured in seconds from the file creation time:

```
Dim series As Long
Dim count As Long
Dim start As Double

AcquireSeriesGetCount session, count

For series = 0 To count - 1
    AcquireSeriesSet session, series
    AcquireSeriesGetStart session, start
Next series
```

A file may have no series if no data was acquired.

5.2.5 Sweeps

A series is a sequence of sweeps. You use the sweep operations to select a sweep and determine its parameters. [☞ Sweep Operations, p. 130.](#)

You use AcquireSweepGetCount to determine the number of sweeps in a file. [☞ AcquireSweepGetCount, p. 152.](#)

Sweeps are numbered 0 to N-1, where N is the number of sweeps in a file. You use AcquireSweepSet to set the sweep to use. [☞ AcquireSweepSet, p. 155.](#)

The following example determines the duration of each sweep in a file. The duration is measured in seconds:

```
Dim sample_rate As Double
Dim sweep_count As Long
Dim current_sweep As Long
Dim duration As Double

AcquireFileGetSampleRate session, sample_rate
AcquireSweepGetCount session, sweep_count
```

```

If sweep_count > 0 Then
  For current_sweep = 0 To sweep_count-1
    AcquireSweepSet session, current_sweep
    AcquireSweepGetDuration session, duration
  Next current_sweep
Else
  ' The file contains no sweeps
End If

```

5.2.6 Segments

A sweep is a sequence of segments. If event screening was not used when a sweep was recorded, the sweep will consist of a single segment. You use the segment operations to select a segment and determine its parameters. [☞ Segment Operations, p. 131.](#)

You use `AcquireSegmentGetCount` to determine the number of segments in a sweep. The number will always be one if event screening was not used during sweep acquisition. [☞ AcquireSegmentGetCount, p. 146.](#)

Segments are numbered 0 to N-1, where N is the number of segments in a sweep. You use `AcquireSegmentSet` to set the segment number to use. [☞ AcquireSegmentSet, p. 148.](#)

The following example determines the starting sample number and number of samples in each segment of the current sweep.

```

Dim segment_count As Long
Dim current_segment As Long
Dim segment_start As Long
Dim samples As Double

AcquireSegmentGetCount session, segment_count

If segment_count > 0 Then
  For current_segment = 0 To segment_count-1
    AcquireSegmentSet session, current_segment
    AcquireSegmentGetStart session, segment_start
    AcquireSegmentGetSampleCount
      session, samples
  Next current_segment
Else
  ' The sweep has no segments
End If

```

A sweep may have no segments if no data was acquired.

5.2.7 Channels

`AcquireChannelGetCount` returns the number of channels recorded in a file. [☞ AcquireChannelGetCount, p. 133.](#)

The following example obtains the number of channels recorded in a file:

```

Dim channel_count As Long

AcquireChannelGetCount session, channel_count

```

Channel numbers range from 0 to N-1, where N is the number of channels in the file.

5.2.8 Control Channels

Markers are associated with control channels. `AcquireControlGetCount` returns the number of control channels recorded in a file. [☞ AcquireControlGetCount, p. 137.](#)

To obtain information about a control channel use `AcquireControlGet`. [☞ AcquireControlGet, p. 136.](#)

The following example iterates through the control channels.

```

Dim name As Variant
Dim signal As Variant
Dim description As Variant
Dim units As Variant
Dim control_count As Long
Dim control As Long

AcquireControlGetCount session, control_count

If control_count > 0 Then
  For control = 0 To control_count - 1
    AcquireControlGet session, control, name,
      description, units, signal

    // Process control channel here.

  Next control
End If

```

5.2.9 Reading Data

Data is read for one channel at a time. If you want to read data for multiple channels, perform multiple read operations.

Reading Segments

To read an entire segment, use `AcquireSegmentRead`.
☞ *AcquireSegmentRead*, p. 147.

The following example reads each segment into an array.

```
Dim buffer() As Double
Dim segment_count As Long
Dim current_segment As Long
Dim samples As Double

AcquireSegmentGetCount(session, segment_count)

If segment_count > 0 Then
  For current_segment = 0 To segment_count-1
    AcquireSegmentSet session, current_segment
    AcquireSegmentGetSampleCount
      session, samples
    ReDim buffer(samples-1)

    AcquireSegmentReadsession, channel, 0, samples,
      buffer(0)
    ' Data processing goes here
  Next current_segment
Else
  ' The sweep has no segments
End If
```

Segments and Channels

Each channel in a segment has the same number of data samples.

The following example reads data from each channel of the current segment into an array.

```
Dim buffer() As Double
Dim channel_count As Long
Dim channel As Long
Dim samples As Double

AcquireSegmentGetSampleCount session, samples
ReDim buffer(samples-1)

AcquireChannelGetCount session, channel_count

For channel = 0 To channel_count-1
  AcquireSegmentRead session, channel, 0, samples,
    buffer(0)
  ' Data processing goes here
Next channel
```

Reading Sections

If a segment is too large to read into memory at one time, you can read sections of a segment. To read a section, use `AcquireSegmentRead`. ☞ *AcquireSegmentRead*, p. 147.

The following example reads each block of the current segment. It creates a single array, the size of the maximum block size. ☞ *AcquireBlockMaxGetSampleCount*, p. 133.

Any processing code inserted in the inner loop may have to obtain the actual number of samples in the current block using `AcquireBlockGetSampleCount`. This is necessary to avoid processing the wrong number of samples in the last block of the segment. ☞ *AcquireBlockGetSampleCount*, p. 132.

```
Dim block_count As Long
Dim block As Long
Dim samples As Long
Dim start As Long
Dim buffer() As Double

AcquireBlockMaxGetSampleCount session, samples
ReDim buffer(samples-1)

AcquireBlockGetCount session, 0, block_count

start = 0

For block = 0 To block_count-1
  AcquireBlockGetSampleCount
    session, block, samples
  AcquireSegmentRead
    session, channel, start, samples, buffer(0)
  ' Data processing goes here
  start = start + samples
Next block
```

5.2.10 Markers

Markers are stored by sweep. To determine the number of markers in the current sweep, use `AcquireMarkerGetCount`.
☞ *AcquireMarkerGetCount*, p. 144.

Each marker has optional associated text and data. To obtain information about a marker use `AcquireMarkerGet`. To obtain the marker text use `AcquireMarkerGetText`.
☞ *AcquireMarkerGet*, p. 143. ☞ *AcquireMarkerGetText*, p. 145.

The following example iterates through the markers in a sweep:

```
Dim control_channel As Long
Dim text As Variant
Dim sample As Double
Dim value As Double
Dim data_size As Long
Dim extended_data_size As Double
Dim marker As Long

AcquireMarkerGetCount session, marker_count, status
marker = 0

If marker_count > 0 Then
  For marker = 0 To marker_count - 1
    AcquireMarkerGet session, marker,
control_channel,
    sample, value, data_size, extended_data_size

    AcquireMarkerGetText session, marker, text

    // Process marker here.
  Next marker
End If
```

5.2.11 Closing a File

To close a file, use `AcquireFileClose`. [☞ AcquireFileClose, p. 138.](#)

The following example closes an Acquire file, then closes the associated session.

```
AcquireFileClose session
AcquireSessionClose session
```

5.2.12 Error Handling

The examples presented in this section take no account of errors that may occur, such as when `AcquireFileOpen` is called for a file that does not contain Acquire data.

If you write procedures to be used by someone else, you will probably be interested in how to handle error conditions in your procedures so you can provide better error recovery for the user. This manual contains information about how to make your procedures more robust. [☞ Robust Processing, p. 129.](#)

5.2.13 Further Information

This manual contains a complete description of each `DataAccess` operation. These descriptions are useful to answer specific questions regarding the individual operations. [☞ Reference, p. 131.](#)

5.3 Robust Processing

If you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

This section explains how to use the information provided by `DataAccess` to recover from error conditions.

5.3.1 Status

All operations return a status value of type `Long`. The value is zero if the operation was successful, and non-zero if an error was detected.

If an error is detected, you can use `AcquireGetStatusText` to translate the status code to a message. [☞ AcquireGetStatusText, p. 142.](#)

The following example shows how the user might be allowed to select a file. The example does not terminate until the user selects a valid file.

```
Dim filespec As Variant
Dim status As Long
Dim session As Long

AcquireSessionOpen session

Do
  ' Add code here to obtain the file specification in "filespec"
  status = AcquireFileOpen session, filespec

  If status = 0 Then
    Exit Do
  End If

  ' Translate the status code to a message here and
  ' display it for the user
Loop
```

This example is not complete, because you should provide

some way for the user to exit the loop without opening a file at all.

5.4 Operations

This chapter describes each of the operations provided by DataAccess. The operations are grouped by category.

5.4.1 Common Operations

The common operations are valid regardless of whether a session is open or not. The common operations are shown in table 72.

Table 72 Common operations

Operation	
AcquireGetVersion	p. 142

5.4.2 Session Operations

The session operations allow you to open and close a session. The session operations are shown in table 73.

Table 73 Session Operations

Operation	
AcquireSessionOpen	p. 151
AcquireSessionClose	p. 151

5.4.3 File Operations

The file operations allow you to access an Acquire data file. The file operations are shown in table 74.

Table 74 File operations

Operation	
AcquireFileOpen	p. 141
AcquireFileClose	p. 138

The file operations are valid when a session is open.

5.4.4 File Parameter Operations

The file parameter operations allow you to obtain file

parameters. The file parameter operations are shown in table 75.

Table 75 File parameter operations

Operation	
AcquireFileGetComment	p. 138
AcquireFileGetSampleRate	p. 139
AcquireFileGetSignature	p. 139
AcquireFileGetTechnique	p. 140
AcquireFileGetTime	p. 140
AcquireFileGetVersion	p. 141

The file parameter operations are valid when a file is open. In this case the returned parameters are the parameters of the open file.

5.4.5 Series Operations

The series operations allow you to access each series of a file. The series operations are shown in table 76.

Table 76 Series operations

Operation	
AcquireSeriesGetCount	p. 149
AcquireSeriesSet	p. 150
AcquireSeriesGetComment	p. 148
AcquireSeriesGetStart	p. 149
AcquireSeriesGetTechnique	p. 150

5.4.6 Sweep Operations

The sweep operations allow you to access each sweep of a file. The sweep operations are shown in table 77.

Table 77 Sweep operations

Operation	
AcquireSweepGetCount	p. 152
AcquireSweepSet	p. 155
AcquireSweepGetComment	p. 152
AcquireSweepGetDuration	p. 153
AcquireSweepGetScreen	p. 153
AcquireSweepGetStart	p. 154
AcquireSweepGetTechnique	p. 154

5.4.7 Segment Operations

The segment operations allow you to access each segment of a sweep. If a file was acquired without event screening, each sweep will have a single segment.

The segment operations are shown in table 78.

Table 78 Segment operations

Operation	
AcquireSegmentGetCount	p. 146
AcquireSegmentSet	p. 148
AcquireSegmentGetSampleCount	p. 146
AcquireSegmentGetStart	p. 147
AcquireSegmentRead	p. 147

5.4.8 Channel Operations

The channel operations allow you to determine the number of channels in a file, and obtain channel parameters. The channel operations are shown in table 79.

Table 79 Channel operations

Operation	
AcquireChannelGetCount	p. 133
AcquireChannelGetDescription	p. 134
AcquireChannelGetRange	p. 134
AcquireChannelGetScaling	p. 135

5.4.9 Block Operations

The block operations allow you to determine the size and number of the blocks within a segment. You normally do not need block operations. If you read a section of a segment, it may be significantly more efficient to read on block boundaries.

The block operations are shown in table 80.

Table 80 Block operations

Operation	
AcquireBlockGetCount	p. 132
AcquireBlockGetSampleCount	p. 132
AcquireBlockMaxGetSampleCount	p. 133

5.4.10 Control Operations

The control operations allow you to determine the number of controls and to obtain the control parameters. The control operations are shown in table 81.

Table 81 Control operations

Operation	
AcquireControlGet	p. 136
AcquireControlGetCount	p. 137
AcquireControlGetHoldingValue	p. 137

5.4.11 Marker Operations

The marker operations allow you to determine the number of markers in the current sweep and to obtain the marker parameters and data. The marker operations are shown in table 82.

Table 82 Marker operations

Operation	
AcquireMarkerGet	p. 143
AcquireMarkerGetCount	p. 144
AcquireMarkerGetData	p. 144
AcquireMarkerGetExtendedData	p. 145
AcquireMarkerGetText	p. 145

5.5 Reference

This section lists each `DataAccess` operation in alphabetical order.

5.5.1 AcquireBlockGetCount

AcquireBlockGetCount returns the number of blocks in the current segment.

Syntax

```
AcquireBlockGetCount(session, channel, count)
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

count A Long variable in which to return the number of blocks in the current segment.

Example

```
Dim block_count As Long

AcquireSweepSet session, 8
AcquireSegmentSet session, 3

AcquireBlockGetCount session, 0, block_count
```

Discussion

You must have a file open, and a current segment selected. To set the current segment, use `AcquireSegmentSet`.
☞ *AcquireSegmentSet*, p. 148.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use `AcquireChannelGetCount`.
☞ *AcquireChannelGetCount*, p. 133.

5.5.2 AcquireBlockGetSampleCount

AcquireBlockGetSampleCount returns the number of samples in the specified block.

Syntax

```
AcquireBlockGetSampleCount
  session, channel, block, samples
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

block A Long value representing the block to use.

samples A Long variable in which to store the number of samples in the current block.

Example

```
Dim samples As Long

AcquireBlockGetSampleCount
  session, channel, 0, samples
```

Discussion

You must have a file open, and a current segment selected. To set the current segment, use `AcquireSegmentSet`.
☞ *AcquireSegmentSet*, p. 148.

For a given channel, all blocks in a file will have the same number of samples, except the last block of each segment, which may have fewer samples.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use `AcquireChannelGetCount`.
☞ *AcquireChannelGetCount*, p. 133.

Blocks are numbered 0 to N-1, where N is the number of blocks in the segment. To determine the number of blocks in the current segment, use `AcquireBlockGetCount`.
☞ *AcquireBlockGetCount*, p. 132.

5.5.3 AcquireBlockMaxGetSampleCount

AcquireBlockMaxGetSampleCount returns the maximum number of samples that are stored in a block.

Syntax

AcquireBlockMaxGetSampleCount *session, samples*

session A Long value representing the current session.

samples A Long variable in which to store the maximum number of samples that are stored in a block.

Example

```
Dim samples As Long
```

```
AcquireBlockMaxGetSampleCount session, samples
```

Discussion

You must have a file open.

For a given channel, all blocks in a file will have this number of samples, except the last block of each segment, which may have fewer samples.

5.5.4 AcquireChannelGetCount

AcquireChannelGetCount returns the number of channels recorded in the current Acquire data file.

Syntax

AcquireChannelGetCount *session, count*

session A Long value representing the current session.

count A Long variable in which to store the number of channels.

Example

```
Dim channel_count As Long
```

```
AcquireChannelGetCount session, channel_count
```

Discussion

The channel count is the number of active channels recorded in the Acquire data file. For example, if data was recorded from rrc16 channels 0, 2, and 5, the channel count is 3. The channel count must be at least 1.

5.5.5 AcquireChannelGetDescription

AcquireChannelGetDescription returns a description of the specified channel.

Syntax

```
AcquireChannelGetDescription session, channel, label,
    signal
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

label A Variant variable in which the channel label is returned.

signal A Variant variable in which the channel signal is returned.

Example

```
Dim label As Variant
Dim signal As Variant
```

```
AcquireChannelGetDescription session, 0, label, signal
```

☞ *Channel Operations, p. 131.*

Discussion

You must have a file open.

The channel label is supplied by Acquire, and is the same as the front-panel label of the channel on the data acquisition device.

The channel signal is supplied by the user.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount. ☞ *AcquireChannelGetCount, p. 133.*

5.5.6 AcquireChannelGetRange

AcquireChannelGetRange returns the range of the specified channel.

Syntax

```
AcquireChannelGetRange session, channel, minimum,
    maximum
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

minimum A Double variable to receive the lower limit.

maximum A Double variable to receive the upper limit.

Example

```
Dim minimum As Double
Dim maximum As Double
```

```
AcquireChannelGetRange session, 0, minimum,
    maximum
```

Discussion

You must have a file open.

AcquireChannelGetRange returns the minimum and maximum possible values for the specified channel. The range of the acquisition device and the user supplied gain are taken into account. ☞ *AcquireChannelGetScaling, p. 135.*

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount. ☞ *AcquireChannelGetCount, p. 133.*

5.5.7 AcquireChannelGetScaling

AcquireChannelGetScaling returns the scaling for the specified channel.

Syntax

```
AcquireChannelGetScaling session, channel, gain, units,
    natural_scale
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

gain A Double variable to receive the gain.

units A Variant variable in which the channel units are returned.

natural_scale A Long variable to receive the scale.

Example

```
Dim gain As Double
Dim scale As Long
Dim units As Variant
```

```
AcquireChannelGetScaling session, 0, gain, units, scale
```

Discussion

You must have a file open.

The channel gain is supplied by Acquire from a value entered by the user.

The channel units are supplied by the user.

The natural scale is a power of 1000 which indicates the natural scale of the measured signal. For example, if the natural scale is pA then AcquireChannelGetNaturalScale will return -4, since $1000^{-4} = 10^{-12}$. The natural scale then corresponds to the standard prefix. Common scalings

are shown in table 83.

Table 83 Common natural scale values

Natural scale	Scale	Prefix
-4	10^{-12}	p (pico)
-3	10^{-9}	n (nano)
-2	10^{-6}	u (micro)
-1	10^{-3}	m (milli)
0	1	
1	10^3	k (kilo)
2	10^6	M (mega)
3	10^9	G (giga)

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use AcquireChannelGetCount. [☞ AcquireChannelGetCount, p. 133.](#)

5.5.8 AcquireControlGet

AcquireControlGet returns the parameters for the specified control.

Syntax

```
AcquireControlGet session, control, name,  
description, units, natural_scale, signal
```

session A Long value representing the current session.

control An int value representing the control index, that is, the number of the control.

name A Variant variable to receive the control name.

description A Variant variable to receive the control description.

units A Variant variable to receive the control value units.

natural_scale A Long variable to receive the scale.

signal A Variant variable to receive the control value name.

Example

```
Dim type As Long  
Dim name As Variant  
Dim description As Variant  
Dim units As Variant  
Dim scale As Long  
Dim signal As Variant
```

```
AcquireControlGet session, 0, name, description, units,  
scale, signal
```

Discussion

You must have a file open.

Controls are numbered 0 to N-1, where N is the number of controls used with the file. To determine the number of controls in the file, use AcquireControlGetCount. [☞ AcquireControlGetCount, p. 137.](#)

The natural scale is a power of 1000 which indicates the natural scale of the control signal. For example, if the natural scale is pA (10^{-12} A), then the natural_scale value

returned will be -4, since $1000^{-4} = 10^{-16}$. The natural scale value corresponds to the standard prefix. Common scalings are shown in table 84. These are not the only scalings that may be supported by Acquire.

Table 84 Common natural scale values

Natural scale	Scale	Prefix
-4	10^{-12}	p (pico)
-3	10^{-9}	n (nano)
-2	10^{-6}	u (micro)
-1	10^{-3}	m (milli)
0	1	
1	10^3	k (kilo)
2	10^6	M (mega)
3	10^9	G (giga)

5.5.9 AcquireControlGetCount

AcquireControlGetCount returns the number of controls used in the current Acquire data file.

Syntax

AcquireControlGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of controls.

Example

```
Dim count As Long
```

```
AcquireControlGetCount session, count
```

Discussion

You must have a file open.

The control count is the number of active controls used with the Acquire data file.

5.5.10 AcquireControlGetHoldingValue

AcquireControlGetHoldingValue returns the holding value for the specified control for the current sweep.

Syntax

AcquireControlGetHoldingValue *session, control, value*

session A Long value representing the current session.

control A Long value representing the control index.

value A Double variable to receive the holding value.

Example

```
Dim value As Double
```

```
AcquireControlGetHoldingValue session, 0, value
```

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 155.](#)

At the end of a sweep, a control output returns to a holding value. The holding value has an associated value to record in the file. This function returns that associated value.

The holding value may vary between sweeps.

Controls are numbered 0 to N-1, where N is the number of controls used with the file. To determine the number of controls in the file, use AcquireControlGetCount. [☞ AcquireControlGetCount, p. 137.](#)

5.5.11 AcquireFileClose

AcquireFileClose terminates processing of an Acquire file.

Syntax

```
AcquireFileClose session
```

session A Long value representing the current session.

Example

```
AcquireFileClose session  
AcquireSessionClose session
```

Discussion

You should perform AcquireFileClose as soon as you complete processing of an Acquire data file. The operation releases the memory required for the Acquire file directory.

5.5.12 AcquireFileGetComment

AcquireFileGetComment returns the comment for the file.

Syntax

```
AcquireFileGetComment session, comment
```

session A Long value representing the current session.

comment A Variant variable in which the file comment is returned.

Example

```
Dim text As Variant  
  
AcquireFileGetComment session, text
```

Discussion

You must have a file open.

5.5.13 AcquireFileGetSampleRate

AcquireFileGetSampleRate returns the number of samples taken per second on each channel.

Syntax

AcquireFileGetSampleRate *session, rate*

session A Long value representing the current session.

rate A Double variable to receive the sampling rate used in the file.

Example

Dim rate As Double

AcquireFileGetSampleRate session, rate

Discussion

You must have a file open.

All channels are sampled at the same rate, and the sampling rate is the same throughout the file.

The value returned is the number of samples taken per second on each channel. If three channels are sampled, the total sampling rate is the sampling rate on each channel times three.

5.5.14 AcquireFileGetSignature

AcquireFileGetSignature returns a unique signature for the data file.

Syntax

AcquireFileGetSignature *session, signature*

session A Long value representing the current session.

signature A Variant variable in which the file signature is returned.

Example

Dim signature As Variant

AcquireFileGetSignature session, signature

Discussion

You must have a file open.

The returned signature is a string containing a 32 character hexadecimal value.

The signature is calculated from information in the file. Except for a very unlikely coincidence, the signature should uniquely identify the file.

5.5.15 AcquireFileGetTechnique

AcquireFileGetTechnique returns the technique string for the file.

Syntax

```
AcquireFileGetTechnique session, technique
```

session A Long value representing the current session.

technique A Variant variable in which the file technique is returned.

Example

```
Dim text As Variant
```

```
AcquireFileGetTechnique session, text
```

Discussion

The technique represents information provided during recording that is intended to be used during data analysis. The technique string can only be set through Acquire scripting. It cannot be set through the Acquire user interface.

You must have a file open.

5.5.16 AcquireFileGetTime

AcquireFileGetTime returns the creation date and time of the file.

Syntax

```
AcquireFileGetTime session, date_time
```

session A Long value representing the current session.

date_time A Date variable to receive the file creation date and time.

Example

```
Dim date_time As Date
```

```
AcquireFileGetTime session, date_time
```

Discussion

You must have a file open.

5.5.17 AcquireFileGetVersion

AcquireFileGetVersion returns the Acquire format version of the file.

Syntax

```
AcquireFileGetVersion session, version
```

session A Long value representing the current session.

version A Variant variable in which the file version is returned.

Example

```
Dim version As Variant
```

```
AcquireFileGetVersion session, version
```

Discussion

You must have a file open.

5.5.18 AcquireFileOpen

AcquireFileOpen prepares an Acquire data file for processing.

Syntax

```
AcquireFileOpen session, path
```

session A Long value representing the current session.

path A Variant value representing the file path.

Example

```
Dim file_name As Variant
file_name = "Data.dat"
AcquireFileOpen session, file_name
```

Discussion

AcquireFileOpen loads the Acquire file directory into memory. This directory describes the location of each sweep and segment in the file, and makes access to any part of the file efficient. The directory can require a substantial amount of memory, so it is possible that AcquireFileOpen will fail due to limited memory.

Once a file is open, you can access file parameters. [☞ File Parameter Operations, p. 130](#). You can process the sweeps in the file. [☞ Sweep Operations, p. 130](#).

When you are done with a file, you should close it. [☞ AcquireFileClose, p. 138](#).

AcquireFileOpen invalidates the current sweep. You must set the current sweep using AcquireSweepSet. [☞ AcquireSweepSet, p. 155](#).

5.5.19 AcquireGetStatusText

AcquireGetStatusText translates a status value returned by a DataAccess operation to a text string.

Syntax

```
AcquireGetStatusText session, status, message
```

session A Long value representing the current session.

status A Long value representing the status value to translate.

message A Variant variable in which the translated text is returned.

Example

```
Dim status As Long
Dim message As Variant

status = AcquireFileOpen(session, file_path)

If status <> 0 Then
    AcquireGetStatusText session, status, message
    ' Display the message here
End If
```

Discussion

AcquireGetStatusText does not return a value, so it should not be called as a function.

AcquireGetStatusText is the only means provided to interpret status values.

5.5.20 AcquireGetVersion

AcquireGetVersion returns the version number of the DataAccess package.

Syntax

```
AcquireGetVersion version
```

version A Long variable to receive the DataAccess version number.

Example

```
Dim version As Long
AcquireGetVersion version
```

Discussion

AcquireGetVersion does not return a value, so it should not be called as a function.

If you are writing a set of procedures using DataAccess, you can use AcquireGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess. You can use the following code to ensure that this version or a later version is in use:

```
Dim version As Long
AcquireGetVersion(version)

If version < N Then
    ' Handle the error
End If
```

5.5.21 AcquireMarkerGet

AcquireMarkerGet returns the parameters for the specified marker.

Syntax

```
AcquireMarkerGet session, marker, control, sample, value,  
data_size, extended_data_size
```

session A Long value representing the current session.

marker An Long value representing the marker for which to return the parameters.

control A Long variable to receive the index of the control which created the marker.

sample A Double variable to receive the sample number from the beginning of the sweep indicating the time of the marker.

value A Double variable to receive the marker value.

data_size A Long variable to receive the size in bytes of the data associated with the marker.

extended_data_size A Long variable to receive the size in bytes of the extended data associated with the marker.

Example

```
Dim control As Long  
Dim value As Double  
Dim sample As Double  
Dim data_size As Long  
Dim extended_data_size As Double  
  
AcquireMarkerGet session, marker, control, sample,  
value, data_size, extended_data_size
```

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 155.](#)

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use AcquireMarkerGetCount. [☞ AcquireMarkerGetCount, p. 144.](#)

The extended data is limited in size only by the memory storage available on your system.

5.5.22 AcquireMarkerGetCount

AcquireMarkerGetCount returns the number of markers contained in the current sweep.

Syntax

AcquireMarkerGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of markers in the current sweep.

Example

```
Dim count As Long
```

```
AcquireMarkerGetCount session, count
```

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 155.](#)

5.5.23 AcquireMarkerGetData

AcquireMarkerGetData returns the data associated with the marker.

Syntax

AcquireMarkerGetDatasection, marker, data, size

session A Long value representing the current session.

marker A Long value representing the marker from which to get the data.

data An array of Bytes to receive the data.

size A Long value representing the size of the data buffer in bytes.

Example

```
Dim buffer(256) As Byte
```

```
AcquireMarkerGetData session, 0, buffer, 256
```

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 155.](#)

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use AcquireMarkerGetCount. [☞ AcquireMarkerGetCount, p. 144.](#)

Use AcquireMarkerGet to determine the size of the data stored with the marker. [☞ AcquireMarkerGet, p. 143.](#)

5.5.24 AcquireMarkerGetExtendedData

AcquireMarkerGetExtendedData returns the extended data associated with the marker.

Syntax

AcquireMarkerGetExtendedData *session, marker, data, size*

session A Long value representing the current session.

marker A Long value representing the marker from which to get the data.

data An array of Bytes to receive the extended data.

size A Long value representing the size of the data buffer in bytes.

Example

```
Dim buffer() As Byte
Dim control As Long
Dim value As Double
Dim sample As Double
Dim data_size As Long
Dim extended_data_size As Double
```

```
AcquireMarkerGet session, marker, control, sample,
value, data_size, extended_data_size
```

```
ReDim buffer(extended_data_size)
```

```
AcquireMarkerGetExtendedData session, 0, buffer,
extended_data_size
```

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use `AcquireSweepSet`. [☞ AcquireSweepSet, p. 155.](#)

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use `AcquireMarkerGetCount`. [☞ AcquireMarkerGetCount, p. 144.](#)

Use `AcquireMarkerGet` to determine the size of the extended data stored with the marker. The extended data is limited in size only by the memory storage available on your system. [☞ AcquireMarkerGet, p. 143.](#)

5.5.25 AcquireMarkerGetText

AcquireMarkerGetText returns the text associated with the marker.

Syntax

AcquireMarkerGetText *session, marker, text*

session A Long value representing the current session.

marker A Long value representing the marker from which to get the data.

text A Variant variable to receive the text.

Example

```
Dim marker_text As Variant
```

```
AcquireMarkerGetText session, 0, marker_text
```

Discussion

You must have a file open, and a current series and sweep selected. To set the current sweep, use `AcquireSweepSet`. [☞ AcquireSweepSet, p. 155.](#)

Markers are numbered 0 to N-1, where N is the number of markers in the current sweep. To determine the number of markers in the current sweep, use `AcquireMarkerGetCount`. [☞ AcquireMarkerGetCount, p. 144.](#)

Use `AcquireMarkerGet` to determine the size of the text stored with the marker. The text will be at most 256 characters in size. [☞ AcquireMarkerGet, p. 143.](#)

5.5.26 AcquireSegmentGetCount

AcquireSegmentGetCount returns the number of segments in the current sweep.

Syntax

AcquireSegmentGetCount *session, count*

session A Long value representing the current session.

count A Long variable in which the number of segments is returned.

Example

```
Dim segment_count As Long

AcquireSweepSet session, 5
AcquireSegmentGetCount session, segment_count
```

Discussion

You must have a file open and a current sweep selected. You can set the current sweep using `AcquireSweepSet`.
☞ *AcquireSweepSet*, p. 155.

You can set the current segment using `AcquireSegmentSet`.
☞ *AcquireSegmentSet*, p. 148.

The segment count is the same for all channels in the sweep.

5.5.27 AcquireSegmentGetSampleCount

AcquireSegmentGetSampleCount returns the number of samples in the current segment.

Syntax

AcquireSegmentGetSampleCount *session, samples*

session A Long value representing the current session.

samples A Double variable in which the number of samples is returned.

Example

```
Dim samples As Double

AcquireSweepSet session, 2
AcquireSegmentSet session, 4

AcquireSegmentGetSampleCount session, samples
```

Discussion

You must have a file open and a segment selected. You can select the current segment using `AcquireSegmentSet`.
☞ *AcquireSegmentSet*, p. 148.

The number of samples in a segment is always an integer value. It is returned in a Double since it may exceed the maximum integer value that can be returned in a Long.

The number of samples in a segment is the same for all channels.

5.5.28 AcquireSegmentGetStart

AcquireSegmentGetStart returns the time of the current segment within the current sweep, measured in samples.

Syntax

```
AcquireSegmentGetStart session, start
```

session A Long value representing the current session.

start A Double variable in which the starting sample number is returned.

Example

```
Dim segment_start As Double
```

```
AcquireSegmentGetStart session, segment_start
```

Discussion

You must have a file open and a segment selected. You can select the current segment using `AcquireSegmentSet`.
☞ *AcquireSegmentSet*, p. 148.

The starting sample number of a segment plus the number of samples in a segment will be less than or equal to the duration of the containing sweep, measured in samples. `AcquireSegmentGetSampleCount` returns the number of samples in the current segment.
☞ *AcquireSegmentGetSampleCount*, p. 146. `AcquireSweepGetDuration` returns the duration of the current sweep.
☞ *AcquireSweepGetDuration*, p. 153.

5.5.29 AcquireSegmentRead

AcquireSegmentRead reads the specified section of a segment of the specified channel.

Syntax

```
AcquireSegmentRead  
session, channel, start, count, data
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

start A Long value representing the starting sample number to read.

count A Long value representing the number of samples to read.

data An array of “Double” to receive the data. The array must be long enough to contain the specified number of samples.

Example

```
Dim samples As Long  
Dim buffer() As Double
```

```
AcquireBlockGetSampleCount session, samples  
ReDim buffer(samples)
```

```
AcquireSegmentRead  
session, channel, 0, samples, buffer(0)
```

Discussion

You must have a file open, and a current segment selected. To set the current segment, use `AcquireSegmentSet`.
☞ *AcquireSegmentSet*, p. 148.

Reading a section is more efficient if it contains exactly one or more blocks. ☞ *Block Operations*, p. 131.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use `AcquireChannelGetCount`.
☞ *AcquireChannelGetCount*, p. 133.

5.5.30 AcquireSegmentSet

AcquireSegmentSet sets the segment number to use within the current sweep.

Syntax

AcquireSegmentSet *session, segment*

session A Long value representing the current session.

segment A Long value representing the segment to use.

Example

AcquireSegmentSet session, 0

Discussion

You must have a file open and a sweep selected. [☞ AcquireSweepSet, p. 155.](#)

If a sweep contains N segments, segment numbers for the sweep are in the range 0 to N-1. You can obtain the number of segments in the current sweep using AcquireSegmentGetCount. [☞ AcquireSegmentGetCount, p. 146.](#)

If a file was recorded without event screening, each sweep contains only one segment, and the only valid segment number is 0.

5.5.31 AcquireSeriesGetComment

AcquireSeriesGetComment returns the comment for the current series.

Syntax

AcquireSeriesGetComment *session, comment*

session A Long value representing the current session.

comment A Variant variable in which the series comment is returned.

Example

Dim text As Variant

AcquireSeriesGetComment session, text

Discussion

You must have a file open and a series selected.

To set the current series, use AcquireSeriesSet. [☞ AcquireSeriesSet, p. 150.](#)

5.5.32 AcquireSeriesGetCount

AcquireSeriesGetCount returns the number of series in the open file.

Syntax

```
AcquireSeriesGetCount session, count
```

session A Long value representing the current session.

count A Long variable in which the number of series is returned.

Example

```
Dim series_count As Long

AcquireSeriesGetCount session, series_count

If series_count = 0 Then
    // Handle an empty file
End If
```

Discussion

You must have a file open.

The series count may be zero. This indicates that no data was recorded in the file.

To set the current series, use AcquireSeriesSet.
 ☞ *AcquireSeriesSet*, p. 150.

5.5.33 AcquireSeriesGetStart

AcquireSeriesGetStart returns the time at which the current series starts within the file, measured in seconds.

Syntax

```
AcquireSeriesGetStart session, start
```

session A Long value representing the current session.

start A Double variable in which the starting sample number is returned.

Example

```
Dim start As Double

AcquireSeriesGetStart session, start
```

Discussion

The start time of a series is measured in seconds since the file was created, not necessarily since the start of data acquisition. The time is not measured to an accuracy greater than one second.

5.5.34 AcquireSeriesGetTechnique

AcquireSeriesGetTechnique returns the technique for the current series.

Syntax

AcquireSeriesGetTechnique *session, technique*

session A Long value representing the current session.

technique A Variant variable in which the series technique is returned.

Example

Dim text As Variant

AcquireSeriesGetTechnique session, text

Discussion

The technique represents information provided during recording that is intended to be used during data analysis. The technique string can only be set through Acquire scripting. It cannot be set through the Acquire user interface.

You must have a file open and a series selected. To set the current series, use AcquireSeriesSet. [☞ AcquireSeriesSet, p. 150.](#)

5.5.35 AcquireSeriesSet

AcquireSeriesSet sets the series number to use.

Syntax

AcquireSeriesSet *session, series*

session A Long value representing the current session.

series A Long value representing the series number to use.

Example

AcquireSeriesSet session, 0

Discussion

You must have a file open.

If a file contains N series, series numbers are in the range 0 to N-1. You can obtain the number of series in the current file using AcquireSeriesGetCount. [☞ AcquireSeriesGetCount, p. 149.](#)

AcquireSeriesSet invalidates the current sweep. You must set the current sweep using AcquireSweepSet. [☞ AcquireSweepSet, p. 155.](#)

5.5.36 AcquireSessionClose

AcquireSessionClose ends the session. It deallocates any resources allocated DataAccess.

Syntax

```
AcquireSessionClose session
```

session A Long value representing the current session.

Example

```
AcquireFileClose(session)

AcquireSessionClose(session)
```

Discussion

AcquireSessionClose does not return a value, so it should not be called as a function.

You should close a session to deallocate any resources allocated to the session.

5.5.37 AcquireSessionOpen

AcquireSessionOpen begins a DataAccess session.

Syntax

```
AcquireSessionOpen session
```

session A Long variable to contain the created session.

Example

```
Dim session As Long
Dim result As Long
Dim file_name As Variant
file_name = "Sample.dat"

AcquireSessionOpen session

result = AcquireFileOpen(session, file_name)
```

Discussion

You must have a session open to use any other DataAccess calls. The handle returned by AcquireSessionOpen is a parameter to all other calls.

You can have as many sessions open simultaneously as you would like. For example, if you want to have two data files open simultaneously, call AcquireSessionOpen twice. Use one handle when accessing one file and the other handle when accessing the other file.

After you are done with the session, you should call AcquireSessionClose. [☞ AcquireSessionClose, p. 151.](#)

5.5.38 AcquireSweepGetComment

AcquireSweepGetComment returns the comment for the current sweep.

Syntax

AcquireSweepGetComment *session, comment*

session A Long value representing the current session.

comment A Variant variable in which the sweep comment is returned.

Example

```
Dim text As Variant
```

```
AcquireSweepGetComment session, text
```

Discussion

You must have a file open and a series and sweep selected. To set the current series, use AcquireSeriesSet. To set the current sweep, use AcquireSweepSet. [☞ AcquireSeriesSet, p. 150.](#) [☞ AcquireSweepSet, p. 155.](#)

5.5.39 AcquireSweepGetCount

AcquireSweepGetCount returns the number of sweeps in the open file.

Syntax

AcquireSweepGetCount *session, count*

session A Long value representing the current session.

count A Long variable in which the number of sweeps is returned.

Example

```
Dim sweep_count As Long
AcquireSweepGetCount session, sweep_count
```

```
If sweep_count = 0 Then
    ' Handle an empty file
End If
```

Discussion

You must have a file open.

The sweep count may be zero. This indicates that no data was recorded in the file.

To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 155.](#)

5.5.40 AcquireSweepGetDuration

AcquireSweepGetDuration returns the duration of the current sweep, measured in samples.

Syntax

AcquireSweepGetDuration *session, duration*

session A Long value representing the current session.

duration A Long variable in which the duration of the current sweep is returned.

Example

```
Dim sweep_duration As Long

AcquireSweepSet session, 0
AcquireSweepGetDuration session, sweep_duration
```

Discussion

You must have a file open and a current sweep selected. To set the current sweep, use AcquireSweepSet. [☞ AcquireSweepSet, p. 155.](#)

If event screening was not used during data acquisition, the sweep will contain only one segment, and the size of that segment will be the same as the duration of the sweep.

If event screening was used during data acquisition, the sweep may contain discontinuous segments. In that case the sweep duration may be greater than the sum of the segment sizes.

5.5.41 AcquireSweepGetScreen

AcquireSweepGetScreen returns a description of the screening used while recording the current sweep.

Syntax

AcquireSweepGetScreen *session, screen*

session A Long value representing the current session.

screen A Variant variable in which the sweep screening description is returned.

Example

```
Dim screen As Variant

AcquireSweepGetScreen session, screen
```

Discussion

You must have a file open and a series and sweep selected. To set the current series, use AcquireSeriesSet. To set the current sweep, use AcquireSweepSet. [☞ AcquireSeriesSet, p. 150.](#) [☞ AcquireSweepSet, p. 155.](#)

5.5.42 AcquireSweepGetStart

AcquireSweepGetStart returns the time at which the current sweep starts within the current series, measured in seconds.

Syntax

```
AcquireSweepGetStart session, start
```

session A Long value representing the current session.

start A Double variable in which the start time of the current sweep is returned.

Example

```
Dim sweep_start As Double
AcquireSweepGetStart session, sweep_start
```

Discussion

The start time of a sweep is measured in seconds since the series was started. The time is not measured to an accuracy greater than one second.

5.5.43 AcquireSweepGetTechnique

AcquireSweepGetTechnique returns the technique used to record the current sweep.

Syntax

```
AcquireSweepGetTechnique session, technique
```

session A Long value representing the current session.

technique A Variant variable in which the sweep technique is returned.

Example

```
Dim technique As Variant
AcquireSweepGetTechnique session, technique
```

Discussion

The technique represents information provided during recording that is intended to be used during data analysis. The technique string can only be set through Acquire scripting. It cannot be set through the Acquire user interface.

You must have a file open and a series and sweep selected. To set the current series, use `AcquireSeriesSet`. To set the current sweep, use `AcquireSweepSet`. [☞ AcquireSeriesSet, p. 150.](#) [☞ AcquireSweepSet, p. 155.](#)

5.5.44 AcquireSweepSet

AcquireSweepSet sets the sweep number to use.

Syntax

AcquireSweepSet *session*, *sweep*

session A Long value representing the current session.

sweep A Long value representing the sweep number to use.

Example

```
AcquireSweepSet session, 0
```

Discussion

You must have a file open.

If a file contains N sweeps, sweep numbers are in the range 0 to N-1. You can obtain the number of sweeps in the current file using AcquireSweepGetCount.
☞ *AcquireSweepGetCount*, p. 152.

AcquireSweepSet invalidates the current segment. You must set the current segment using AcquireSegmentSet.
☞ *AcquireSegmentSet*, p. 148.

6 PatchMaster: IGOR Pro

Section	
6.1 Script Interface	P. 156
6.2 Robust Processing	P. 158
6.3 Operations	P. 159
6.4 Reference	P. 161

6.1 Script Interface

This section explains how to access a PatchMaster data file. It contains a step by step description of the operations to perform to open a file and read the data in it.

6.1.1 Opening a Data Set

Open an PatchMaster data set using the `PMFileOpen` operation. [↗ *PMFileOpen*, p. 174.](#)

In order to open a data set, you must supply the full path to the `DAT` file. The IGOR PRO “Open” command can be used to bring up a file selection dialog and supply the resulting file path.

The following example brings up a file selection dialog to obtain a file specification. It then uses `PMFileOpen` to open the data set for processing:

```
Variable fileReferenceNumber
Variable status

Open /D/R/T="BINA" fileReferenceNumber

if (strlen(S_fileName) > 0)
    PMFileOpen S_fileName, status
endif
```

6.1.2 File Parameters

Once you have opened a data set, you can determine the parameters of the data set. The parameters include the creation date and time and the data set comment. [↗ *File Parameter Operations*, p. 160.](#)

The following example obtains data set parameters:

```
Variable status
String fileDateTime
PMFileGetTime fileDateTime, status
Print fileDateTime
```

```
String fileComment
PMFileGetComment fileComment, status
Print fileComment
```

6.1.3 Groups

A data set is a sequence of groups. You use the group operations to select a group and determine its parameters. [↗ *Group Operations*, p. 160.](#)

You use `PMGroupGetCount` to determine the number of groups in a data set. [↗ *PMGroupGetCount*, p. 176.](#)

Groups are numbered 1 to N, where N is the number of groups in a data set. You use `PMGroupSet` to set the group to use. [↗ *PMGroupSet*, p. 178.](#)

The following example prints the label of each group in a data set. The label is a text string.

```
Variable group_count
Variable current_group
String group_label
Variable status
```

```
PMGroupGetCount group_count, status
```

```
if (group_count > 0)
    current_group = 1
```

```

do
  PMGroupSet current_group, status
  PMGroupGetLabel group_label, status
  Print group_label
  current_group += 1
  while (current_group <= group_count)
else
  | The data set contains no groups
endif

```

A file may have no groups if no data was acquired.

6.1.4 Series

A group is a sequence of series. You use the series operations to select a series and determine its parameters. [☞ Series Operations, p. 160.](#)

You use PMSeriesGetCount to determine the number of series in a group. [☞ PMSeriesGetCount, p. 181.](#)

Series are numbered 1 to N, where N is the number of series in a group. You use PMSeriesSet to set the series to use. [☞ PMSeriesSet, p. 186.](#)

The following example prints the label of each series in the current group. The label is a text string.

```

Variable series_count
Variable current_series
String series_label
Variable status

PMSeriesGetCount series_count, status

if (series_count > 0)
  current_series = 1
  do
    PMSeriesSet current_series, status
    PMSeriesGetLabel series_label, status
    Print series_label
    current_series += 1
  while (current_series <= series_count)
else
  | The group contains no series
endif

```

A group may have no series if no data was acquired.

6.1.5 Sweeps

A series is a sequence of sweeps. You use the sweep operations to select a sweep and determine its parameters. [☞ Sweep Operations, p. 160.](#)

You use PMSweepGetCount to determine the number of sweeps in a sweep. [☞ PMSweepGetCount, p. 191.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in a series. You use PMSweepSet to set the sweep to use. [☞ PMSweepSet, p. 199.](#)

The following example prints the number of samples in each sweep in the current series.

```

Variable sweep_count
Variable current_sweep
Variable sweep_samples
Variable status

PMSweepGetCount sweep_count, status

if (sweep_count > 0)
  current_sweep = 1
  do
    PMSweepSet current_sweep, status
    PMSweepGetSampleCount, 0,
      sweep_samples, status
    Print sweep_samples
    current_sweep += 1
  while (current_sweep <= sweep_count)
else
  | The series contains no sweep
endif

```

A series may have no sweep if no data was acquired.

6.1.6 Segments

A stimulus template is described by a sequence of segments. Although all sweeps in a series use the same stimulus template, the duration and voltage of each segment can vary with each sweep.

Segment numbers range from 1 to N, where N is the number of segments in the series. You use PMSegmentGetCount to determine the number of segments in a series. [☞ PMSegmentGetCount, p. 179.](#)

6.1.7 Channels

PMChannelGetCount returns the number of channels recorded in a sweep. Normally the number of channels recorded in each sweep in a series will be the same when a data set is acquired, but channels can be removed from individual sweeps during data set editing, leaving a series with varying numbers of channels recorded in each sweep. [☞ PMChannelGetCount, p. 165.](#)

The following example obtains the number of channels recorded in the current sweep:

```
Variable channel_count
Variable status

PMChannelGetCount channel_count, status
```

Channel numbers range from 0 to N-1, where N is the number of channels in the sweep. Existing versions of PatchMaster never record more than two channels in a sweep.

6.1.8 Reading Data

Normally one would read whole sweeps but it may be necessary to read sections of a sweep if you have insufficient memory available to read whole sweeps.

The wave into which you read data must contain double-precision values.

Reading Sweeps

To read an entire sweep, use PMSweepRead. [☞ PMSweepRead, p. 197.](#)

The following example reads each sweep of a series into a wave.

```
Variable sweep_count
Variable current_sweep
Variable samples
Variable status

PMSweepGetCount sweep_count, status

if (sweep_count > 0)
  current_sweep = 1
  do
    PMSweepSet current_sweep, status
```

```
PMSweepGetSampleCount channel,
  samples, status
Make/D/N=(samples) data

PMSweepRead channel, 0, samples, data,
  status
  | Data processing goes here
  current_sweep += 1
  while (current_sweep <= sweep_count)
else
  | The series has no sweeps
endif
```

Reading Sections of a Sweep

To read sections of a sweep, use PMSweepRead. [☞ PMSweepRead, p. 197.](#)

The following example reads the first “length” values of the first sweep into a wave.

```
Variable samples
Variable status

PMSweepGetSampleCount 0, samples, status

if (samples > length)
{
  samples = length;
};

Make/D/N=(samples) data

PMSweepRead channel, 0, samples, data, status
```

6.1.9 Closing a Data Set

To close a data set, use PMFileClose. [☞ PMFileClose, p. 171.](#)

The following example closes a PatchMaster data set.

```
PMFileClose
```

6.2 Robust Processing

If you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

This section explains how to use the information provided by DataAccess Pro to recover from error conditions in your IGOR Pro procedures.

6.2.1 Errors

The status information provided by DataAccess Pro allows you to recover from error conditions caused by the program, user or the system. It does not allow you to recover from syntax errors in your program.

Program Errors

Errors in your program include such mistakes as using PMSeriesSet to set an invalid series number. You can determine the range of valid series numbers using PMSeriesGetCount, so you can avoid setting an invalid segment number. Catching and reporting such errors is helpful in debugging a procedure.

System Errors

System errors include such problems as an error reading a PatchMaster data file. Such errors are usually the result of problems such as a corrupted disk, a missing floppy or CD-ROM, or defective hardware such as a disk drive. Catching such errors prevents further problems and may allow the procedure to recover, for example by allowing the user to insert a missing file medium.

User Errors

User errors include mistakes such as when the user selects a file that is not a PatchMaster data file. The recovery allows your procedures to provide an error indication to the user and allow the user to select a different file.

Syntax Errors

Syntax errors include mistakes such as passing the wrong number or types of parameters to an operation. An improperly dimensioned wave is also treated as a syntax error. Such errors are returned directly to Igor Pro and cause a procedure to terminate.

6.2.2 Status

Nearly all operations return a status value through a status

parameter. The status value is non-zero if an error occurred. All Data Access and system errors are returned in this manner. Such errors will not terminate a procedure. The procedure should be designed to handle such conditions. Igor Pro errors, such as a missing parameter, are returned directly to Igor and will terminate a procedure.

Status values for specific errors are not defined, and may change from version to version of DataAccess Pro. The PMGetStatusText operation translates a status value to text. *PMGetStatusText*, p. 175.

The following example allows the user to select a file, and does not terminate until the user selects a valid PatchMaster file.

```

Variable fileName
Variable status
String message

do
  Open /D/R/T="BINA" fileName

  if (strlen(S_fileName) > 0)
    PMFileOpen S_fileName, status

    if (status != 0)
      PMGetStatusText status, message
      Print "PMFileOpen error: ", message
    endif
  endif
while (status != 0)

```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

6.3 Operations

This chapter describes each of the operations provided by DataAccess Pro. The operations are grouped by category.

6.3.1 Common Operations

The common operations are valid regardless of whether a data set is open or not. The common operations are shown

in table 85.

Table 85 Common operations

Operation	
PMGetVersion	p. 175
PMGetStatusText	p. 175

6.3.2 File Operations

The file operations allow you to access a PatchMaster data set. The file operations are shown in table 86.

Table 86 File operations

Operation	
PMFileOpen	p. 174
PMFileClose	p. 171

6.3.3 File Parameter Operations

The file parameter operations allow you to obtain data set parameters. The file parameter operations are shown in table 87.

Table 87 File parameter operations

Operation	
PMFileGetComment	p. 172
PMFileGetMarked	p. 172
PMFileGetTime	p. 173
PMFileGetPMVersion	p. 173
PMFileGetVersion	p. 174

The file parameter operations are valid when a data set is open.

6.3.4 Group Operations

The group operations allow you to access each group of a data set. The group operations are shown in table 88.

Table 88 Group operations

Operation	
PMGroupGetComment	p. 176
PMGroupGetCount	p. 176
PMGroupGetExperiment	p. 177
PMGroupGetLabel	p. 177

Table 88 Group operations

Operation	
PMGroupGetMarked	p. 178
PMGroupSet	p. 178

The group operations are valid when a data set is open. Most require that a current group be set.

6.3.5 Series Operations

The series operations allow you to access each series of a group. The series operations are shown in table 89.

Table 89 Series operations

Operation	
PMSeriesGetComment	p. 181
PMSeriesGetCount	p. 181
PMSeriesGetLabel	p. 182
PMSeriesGetLeak	p. 182
PMSeriesGetMarked	p. 183
PMSeriesGetSampleInterval	p. 184
PMSeriesGetStart	p. 184
PMSeriesGetTime	p. 185
PMSeriesGetUserParameter	p. 186
PMSeriesSet	p. 186

The series operations are valid when a data set is open. Most require that a current series be set.

6.3.6 Sweep Operations

The sweep operations allow you to access each sweep of a series. The sweep operations are shown in table 90.

Table 90 Sweep operations

Operation	
PMSweepGetCount	p. 191
PMSweepGetDigitalIn	p. 191
PMSweepGetHoldingPotential	p. 192
PMSweepGetLabel	p. 192
PMSweepGetMarked	p. 193
PMSweepGetRecordedCount	p. 193
PMSweepGetSampleCount	p. 194
PMSweepGetSolutions	p. 194
PMSweepGetStart	p. 195
PMSweepGetTemperature	p. 195

Table 90 Sweep operations

Operation	
PMsweepGetTimer	p. 196
PMsweepGetUserParameter	p. 196
PMsweepRead	p. 197
PMsweepReadLeak	p. 197
PMsweepReadRaw	p. 198
PMsweepReadStimulus	p. 198
PMsweepSet	p. 199

The sweep operations are valid when a data set is open. Most require that a current sweep be set.

6.3.7 Channel Operations

The channel operations allow you to obtain information about the acquired data channels. The channel operations are shown in table 91.

Table 91 Channel operations

Operation	
PMChannelGetADC	p. 162
PMChannelGetAmplifier	p. 162
PMChannelGetAvailable	p. 163
PMChannelGetBandwidth	p. 163
PMChannelGetCellPotential	p. 164
PMChannelGetCompression	p. 164
PMChannelGetCount	p. 165
PMChannelGetLabel	p. 166
PMChannelGetLeakAvailable	p. 166
PMChannelGetLinkedStimulus	p. 167
PMChannelGetMarked	p. 167
PMChannelGetPipetteResistance	p. 168
PMChannelGetRange	p. 168
PMChannelGetSampleInterval	p. 169
PMChannelGetSealResistance	p. 170
PMChannelGetStart	p. 170
PMChannelGetUnits	p. 171

6.3.8 Stimulus Operations

The stimulus operations allow you to obtain information about the stimulus protocol. The stimulus operations are

shown in table 92.

Table 92 Stimulus operations

Operation	
PMStimulusGetCount	p. 187
PMStimulusGetDAC	p. 187
PMStimulusGetRange	p. 188
PMStimulusGetRelevantSegments	p. 188
PMStimulusGetSource	p. 189
PMStimulusGetStartSegment	p. 189
PMStimulusGetTriggerType	p. 190
PMStimulusGetUnits	p. 190

6.3.9 Segment Operations

The segment operations allow you to access information about the stimulus segments. The segment operations are shown in table 93.

Table 93 Segment operations

Operation	
PMSegmentGetCount	p. 179
PMSegmentGetParameters	p. 179

6.4 Reference

This section lists each DataAccess Pro operation in alphabetical order.

6.4.1 PMChannelGetADC

PMChannelGetADC obtains the physical A/D channel.

Syntax

PMChannelGetADC *channel, adc, status*

channel A value representing the channel.

adc A variable to receive the A/D channel number.

status A variable to receive the status of the operation.

Example

Variable adc
Variable status

```
PMChannelGetADC 0, adc, status
```

Discussion

The logical channel number may differ from the physical A/D channel used. The physical channel number is the one labeled on the data acquisition device.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [☞ PMChannelGetCount, p. 165.](#)

The current series must be set. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

6.4.2 PMChannelGetAmplifier

PMChannelGetAmplifier returns the amplifier settings for the current sweep for the specified channel.

Syntax

PMChannelGetAmplifier *channel, adc*

channel A value representing the channel

gain A variable to receive the gain.

capacitance A variable to receive the capacitance.

conductance A variable to receive the series conductance.

resistance A variable to receive the series resistance compensation.

status A variable to receive the status of the operation.

Example

Variable status
Variable gain
Variable capacitance
Variable conductance
Variable resistance

```
PMChannelGetAmplifier channel,  
gain, capacitance, conductance, resistance, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [☞ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [☞ PMSweepSet, p. 199.](#)

6.4.3 PMChannelGetAvailable

PMChannelGetAvailable returns whether the channel trace is available for the current sweep.

Syntax

```
PMChannelGetAvailable channel,
    available, status
```

channel A value representing the channel.

available A variable to receive the available status.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable available
```

```
PMChannelGetAvailable channel, available,
    status
```

```
if (available != 0)
    | Process the data
endif
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

It is possible to edit data files in PatchMaster, so a trace for a given sweep can be deleted from the data file.

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.4 PMChannelGetBandwidth

PMChannelGetBandwidth returns the bandwidth for the specified channel for the current sweep.

Syntax

```
PMChannelGetBandwidth channel, bandwidth,
    status
```

channel A value representing the channel.

bandwidth A variable to receive the bandwidth.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable bandwidth
```

```
PMChannelGetBandwidth channel,
    bandwidth, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.5 PMChannelGetCellPotential

PMChannelGetCellPotential returns the cell potential for the specified channel for the current sweep.

Syntax

```
PMChannelGetCellPotential channel,
    cell_potential, status
```

channel A value representing the channel.

cell_potential A variable to receive the cell potential.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable cell_potential
```

```
PMChannelGetCellPotential channel,
    cell_potential, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [☞ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [☞ PMSweepSet, p. 199.](#)

6.4.6 PMChannelGetCompression

PMChannelGetCompression returns the data compression applied to the specified channel.

Syntax

```
PMChannelGetCompression channel,
    factor, skip, averaged, filtered, status
```

channel A value representing the channel to use.

factor A variable to receive the decimation factor.

skip A variable to receive the start offset.

averaged A variable to receive the averaging mode.

filtered A variable to receive the filtered mode.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable factor
Variable skip
Variable averaged
Variable filtered
```

```
PMChannelGetCompression 0, factor,
    skip, averaged, filtered, status
```

Discussion

PMChannelGetCompression returns the compression parameters for the specified channel. The data is decimated by the factor. The decimation can start at an offset returned by the skip parameter. The offset will always be less than decimation factor.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [☞ PMChannelGetCount, p. 165.](#)

If averaged is non-zero, the samples are averaged together, otherwise single samples are used.

If filtered is non-zero, a digital filter has been applied to the data.

The current series must be set. To set the current series, use `PMSeriesSet`. [↗ *PMSeriesSet*, p. 186.](#)

6.4.7 PMChannelGetCount

`PMChannelGetCount` returns the number of channels recorded in the current sweep.

Syntax

`PMChannelGetCount count, status`

count A variable to receive the number of channels recorded in the current sweep.

status A variable to receive the status of the operation.

Example

```
Variable status  
Variable channels
```

```
PMChannelGetCount channels, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series.

The current series must be set. To set the current series, use `PMSeriesSet`. [↗ *PMSeriesSet*, p. 186.](#)

6.4.8 PMChannelGetLabel

PMChannelGetLabel returns the label for the specified channel.

Syntax

PMChannelGetLabel *channel, label, status*

channel A value representing the channel.

label A String variable to receive the label.

status A variable to receive the status of the operation.

Example

Variable status
String label

```
PMChannelGetLabel channel, label,
status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.9 PMChannelGetLeakAvailable

PMChannelGetLeakAvailable returns whether the channel's leak trace is available for the current sweep.

Syntax

PMChannelGetLeakAvailable *channel, available, status*

channel A Long value representing the channel.

available An Long variable to receive the available status.

status A variable to receive the status of the operation.

Example

Variable status
Variable available

```
PMChannelGetLeakAvailable ,channel,
available, status
```

```
if (available != 0)
| Process the leak data
endif
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

It is possible to edit data files in PatchMaster, so a leak trace for a given sweep can be deleted from the data file.

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.10 PMChannelGetLinkedStimulus

PMChannelGetLinkedStimulus returns the stimulus channel associated with the specified channel.

Syntax

```
PMChannelGetLinkedStimulus channel,
    linked_channel, status
```

channel A Long value representing the channel.

linked_channel An Long variable to receive the linked stimulus channel number.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable linked_channel
```

```
PMChannelGetLinkedStimulus channel, linked_channel,
status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

6.4.11 PMChannelGetMarked

PMChannelGetMarked returns whether the specified channel has been marked.

Syntax

```
PMChannelGetMarked channel, marked, status
```

channel A Long value representing the channel.

marked An Long variable to receive the marked state.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable marked
```

```
PMChannelGetMarked channel, marked, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.12 PMChannelGetPipetteResistance

PMChannelGetPipetteResistance returns the pipette resistance for the specified channel.

Syntax

```
PMChannelGetPipetteResistance channel,
    resistance
```

channel A Long value representing the channel to use.

resistance A Double variable to receive the pipette resistance.

status A variable to receive the status of the operation.

Example

```
Variable count
Variable resistance
```

```
PMChannelGetPipetteResistance channel,
    resistance, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.13 PMChannelGetRange

PMChannelGetRange returns the range of the specified channel.

Syntax

```
PMChannelGetRange channel, minimum, maximum,
    status
```

channel A value representing the channel to use.

minimum A variable to receive the lower limit.

maximum A variable to receive the upper limit.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable minimum
Variable maximum
```

```
PMChannelGetRange 0, minimum, maximum, status
```

Discussion

PMChannelGetRange returns the minimum and maximum possible values for the specified channel.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.14 PMChannelGetRecordingMode

PMChannelGetRecordingMode returns the recording mode for the specified channel.

Syntax

```
PMChannelGetRecordingMode channel,
mode, status
```

channel A value representing the channel.

mode A variable to receive the recording mode.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable mode
```

```
PMChannelGetRecordingMode channel, mode,
status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [☞ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [☞ PMSweepSet, p. 199.](#)

The recording mode is one of the values listed in table 94.

Table 94 Recording modes

Mode	Protocol
0	In out
1	On cell
2	Out out
3	Whole cell
4	C clamp
5	V clamp

6.4.15 PMChannelGetSampleInterval

PMChannelGetSampleInterval returns the sampling interval in seconds for the specified channel.

Syntax

```
PMChannelGetSampleInterval channel, interval,
status
```

channel A value representing the channel.

interval A variable to receive the sampling interval.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable interval
```

```
PMChannelGetSampleInterval channel,
interval, status
```

Discussion

All channels are sampled at the same interval. The interval measures the time between samples on the same channel. However a given channel may be compressed, resulting in a larger interval than for the series. [☞ PMSeriesGetSampleInterval, p. 184.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [☞ PMChannelGetCount, p. 165.](#)

The current series must be set. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

6.4.16 PMChannelGetSealResistance

PMChannelGetSealResistance returns the seal resistance for the specified channel.

Syntax

```
PMChannelGetSealResistance channel, resistance,
    status
```

channel A Long value representing the channel.

resistance A Double variable to receive the seal resistance.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable resistance
```

```
PMChannelGetSealResistance channel,
    resistance, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.17 PMChannelGetStart

PMChannelGetStart returns the precise start time in seconds relative to the stimulus start for the specified channel.

Syntax

```
PMChannelGetStart channel, resistance, status
```

channel A Long value representing the channel.

time A Double variable to receive the start time.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable time
```

```
PMChannelGetStart channel, time, status
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [↗ PMChannelGetCount, p. 165.](#)

The current sweep must be set. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

The start time may include offsets due to a nonzero start segment and time set for the stimulus protocol. It may also include sampling sequence delays.

6.4.18 PMChannelGetUnits

PMChannelGetUnits returns the units associated with the specified channel.

Syntax

PMChannelGetUnits *channel, units, status*

channel A value specifying the channel for which to return the units.

units A String variable to contain the units.

status A variable to receive the status of the operation.

Example

Variable status

String units

PMChannelGetUnits channel, units, status

Discussion

The units for a given channel channels recorded may vary from series to series within a group.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PMChannelGetCount. [☞ PMChannelGetCount, p. 165.](#)

The current series must be set. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

6.4.19 PMFileClose

PMFileClose terminates processing of the PatchMaster data set.

Syntax

PMFileClose

status A variable to receive the status of the operation.

Example

PMFileClose

Discussion

You must have a data set open.

You should perform PMFileClose as soon as you complete processing of a data set.

6.4.20 PMFileGetComment

PMFileGetComment returns the comment for the data set.

Syntax

PMFileGetComment *comment, status*

comment A String variable in which the data set comment is returned.

status A variable to receive the status of the operation.

Example

Variable status
String text

PMGetFileCommenttext, status

Discussion

You must have a data set open.

6.4.21 PMFileGetMarked

PMFileGetMarked returns whether the file has been marked.

Syntax

PMFileGetMarked *channel, marked, status*

marked An Long variable to receive the marked state.

status A variable to receive the status of the operation.

Example

Variable status
Variable marked

PMFileGetMarked channel, marked, status

Discussion

You must have a data set open.

6.4.22 PMFileGetTime

PMFileGetTime returns the creation date and time of the data set.

Syntax

PMFileGetTime *date_time, status*

date_time A String variable to receive the data set creation date and time.

status A variable to receive the status of the operation.

Example

Variable status
String date_time

PMFileGetTime date_time, status

Discussion

You must have a data set open.

6.4.23 PMFileGetPMVersion

PMFileGetPMVersion returns the version of PatchMaster which created the file.

Syntax

PMFileGetPMVersion *version, status*

version A String variable to receive the PatchMaster version.

status A variable to receive the status of the operation.

Example

Variable status
String version

PMFileMasterGetPMVersion version, status

Discussion

You must have a data set open.

6.4.24 PMFileGetVersion

PMFileGetVersion returns the PatchMaster file format version of the file.

Syntax

PMFileGetVersion *version, status*

version A String variable to receive the file version number.

status A variable to receive the status of the operation.

Example

Variable status

String version

PMFileGetVersion version, status

Discussion

You must have a data set open.

6.4.25 PMFileOpen

PMFileOpen prepares a PatchMaster data set for processing.

Syntax

PMFileOpen *path, status*

path A String value representing the data set path.

status A variable to receive the status of the operation.

status A variable to receive the status of the operation.

Example

Variable status

PMFileOpen "Data.dat", status

Discussion

You must not have a data set open.

A PatchMaster data set consists of three files, with the extensions PUL, PGF, and DAT. All three files must be present in the same directory for PMFileOpen to succeed.

Once a data set is open, you can access data set parameters. [☞ File Parameter Operations, p. 160](#). You can process the groups in the data set. [☞ Group Operations, p. 160](#).

When you are done with a data set, you should close it. [☞ PMFileClose, p. 171](#).

PMFileOpen invalidates the current group. You must set the current group using PMGroupSet. [☞ PMGroupSet, p. 178](#).

6.4.26 PMGetStatusText

PMGetStatusText translates a status value returned by a DataAccess Pro operation to a text string.

Syntax

PMGetStatusText *status, message*

status A value representing the status value to translate.

message A String variable in which the translated text is returned.

status A variable to receive the status of the operation.

Example

Variable status
String message

PMFileOpen "Sample.dat", status

```
if (status <> 0)
  PMGetStatusText status, message
  Print "PMFileOpen error: ", message
endif
```

Discussion

You need not have a data set open.

PMGetStatusText is the only means provided to interpret status values.

6.4.27 PMGetVersion

PMGetVersion returns the version number of the DataAccess Pro package.

Syntax

PMGetVersion *version*

version A variable to receive the DataAccess Pro version number.

status A variable to receive the status of the operation.

Example

Variable version

PMGetVersion version

Discussion

You need not have a data set open.

If you are writing a set of procedures using DataAccess Pro, you can use PMGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess Pro. You can use the following code to ensure that this version or a later version is in use:

Variable version
PMGetVersion version

```
if (version < N)
  | Handle the error
endif
```

6.4.28 PMGroupGetComment

PMGroupGetComment returns the comment for the current group.

Syntax

PMGroupGetComment *comment, status*

comment A String variable to receive the group comment.

status A variable to receive the status of the operation.

Example

Variable status
String comment

PMGroupGetComment comment, status

Discussion

The group comment is entered by the user, and may be empty.

To set the current group, use PMGroupSet.
☞ *PMGroupSet*, p. 178.

6.4.29 PMGroupGetCount

PMGroupGetCount returns the number of groups in the data set.

Syntax

PMGroupGetCount *count, status*

count A variable to receive the number of groups in the current data set.

status A variable to receive the status of the operation.

Example

Variable status
Variable group_count

PMGroupGetCount count, status

Discussion

You must have a data set open.

6.4.30 PMGroupGetExperiment

PMGroupGetExperiment returns the group experiment number.

Syntax

PMGroupGetExperiment *experiment, status*

experiment A variable to receive the group experiment number.

status A variable to receive the status of the operation.

Example

Variable status
Variable group_experiment

PMGroupGetExperiment group_experiment, status

Discussion

The current group must be set. To set the current group, use PMGroupSet. [☞ PMGroupSet, p. 178.](#)

6.4.31 PMGroupGetLabel

PMGroupGetLabel returns the label for the current group.

Syntax

PMGroupGetLabel *label, status*

label A String variable to receive the group label.

status A variable to receive the status of the operation.

Example

Variable status
String group_label

PMGroupGetLabel group_label, status

Discussion

To set the current group, use PMGroupSet. [☞ PMGroupSet, p. 178.](#)

6.4.32 PMGroupGetMarked

PMGroupGetMarked returns whether the current group has been marked.

Syntax

PMGroupGetMarked *marked, status*

marked An Long variable to receive the marked state.

status A variable to receive the status of the operation.

Example

Variable status
Variable marked

PMGroupGetMarked marked, status

Discussion

The current group must be set. To set the current group, use PMGroupSet. [☞ PMGroupSet, p. 178.](#)

6.4.33 PMGroupSet

PMGroupSet selects the current group.

Syntax

PMGroupSet *group, status*

group A value representing the group to select.

status A variable to receive the status of the operation.

Example

The following example sets the current group to the last group in a data set.

Variable status
Variable group_count

PMGroupGetCount group_count, status

PMGroupSet group_count, status

Discussion

You must have a data set open.

The groups in a data set are numbered from 1 to N, where N is the value returned by PMGroupGetCount. [☞ PMGroupGetCount, p. 176.](#)

6.4.34 PMSegmentGetCount

PMSegmentGetCount returns the number of segments for the specified stimulus channel.

Syntax

PMSegmentGetCount *channel, count, status*

channel A value representing the stimulus channel to use.

count A variable to receive the number of stimulus segments in the current series.

status A variable to receive the status of the operation.

Example

Variable status
Variable segment_count

PMSegmentGetCount channel, count, status

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PMStimulusGetCount. [☞ PMStimulusGetCount, p. 187.](#)

The current series must be set. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

6.4.35 PMSegmentGetParameters

PMSegmentGetParameters returns the parameters of the specified segment for the current sweep.

Syntax

PMSegmentGetParameters *channel, segment, type, voltage, duration, status*

channel A Long value representing the stimulus channel to use.

segment A value representing the segment to use.

type A variable to receive the type of the segment.

voltage A variable to receive the voltage of the segment.

duration A variable to receive the duration of the segment. The duration is measured in samples.

status A variable to receive the status of the operation.

Example

The following example obtains the parameters of the last segment of the current series.

Variable status
Variable segment
Variable segment_type
Variable voltage
Variable duration

PMSegmentGetCount channel, segment, status

```
if (segment < 1)
  | Handle the error
endif
```

PMSegmentGetParameters channel, segment,
segment_type, voltage, duration, status

Discussion

The segment type value is interpreted according to

table 95.

Table 95 Segment types

Type	Interpretation
0	Constant
1	Ramp
2	Continuous
3	Sine wave
4	Square wave

If the segment type is constant or continuous, the voltage represents a fixed output value.

If the segment type is ramp, the voltage represents the final voltage of the segment. The initial value is the final voltage of the previous segment.

If the segment type is either sine or square wave, the output is a constant value of specified voltage plus a sine wave or square wave waveform. The parameters of the waveform may be obtained using `PMSegmentGetWaveform`.
 ☞ *PMSegmentGetWaveform*, p. 180.

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use `PMStimulusGetCount`.
 ☞ *PMStimulusGetCount*, p. 187.

You must have a sweep selected. To set the current sweep, use `PMSweepSet`. ☞ *PMSweepSet*, p. 199.

Segment numbers are in the range 1 to N, where N is the value returned by `PMSegmentGetCount`.
 ☞ *PMSegmentGetCount*, p. 179.

6.4.36 PMSegmentGetWaveform

`PMSegmentGetWaveform` returns the periodic waveform parameters for the specified stimulus channel.

Syntax

```
PMSegmentGetWaveform channel, period,
voltage, status
```

channel A Long value representing the stimulus channel to use.

period A Double variable to receive the waveform period, measured in seconds.

voltage A Double variable to receive the waveform voltage, measured in volts.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable period
Variable voltage
```

```
PMSegmentGetWaveform channel, period,
voltage, status
```

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use `PMStimulusGetCount`.
 ☞ *PMStimulusGetCount*, p. 187.

The current series must be set. To set the current series, use `PMSeriesSet`. ☞ *PMSeriesSet*, p. 186.

The waveform parameters apply if the segment type is sine wave or square wave. You can use `PMSegmentGetParameters` to obtain the type of a segment.
 ☞ *PMSegmentGetParameters*, p. 179.

6.4.37 PMSeriesGetComment

PMSeriesGetComment returns the comment for the current series.

Syntax

PMSeriesGetComment *comment, status*

comment A String variable to receive the series comment.

status A variable to receive the status of the operation.

Example

Variable status
String comment

PMSeriesGetComment comment, status

Discussion

The series comment is entered by the user, and may be empty.

The current series must be set. To set the current series, use PMSeriesSet. ↗ *PMSeriesSet*, p. 186.

6.4.38 PMSeriesGetCount

PMSeriesGetCount returns the number of series in the current group.

Syntax

PMSeriesGetCount *count, status*

count A variable to receive the number of series in the current group.

status A variable to receive the status of the operation.

Example

Variable status
Variable series_count

PMSeriesGetCount count, status

Discussion

The current group must be set. To set the current group, use PMGroupSet. ↗ *PMGroupSet*, p. 178.

6.4.39 PMSeriesGetLabel

PMSeriesGetLabel returns the label for the current series.

Syntax

PMSeriesGetLabel *label, status*

label A String variable to receive the series label.

status A variable to receive the status of the operation.

Example

Variable status

String label

PMSeriesGetLabel label, status

Discussion

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

6.4.40 PMSeriesGetLeak

PMSeriesGetLeak returns the leak parameters for the current series.

Syntax

PMSeriesGetLeak *count, scaling, potential, alternate, alternate_averaging, delay, status*

count A variable to receive the number of sweeps averaged for a leak trace.

scaling A variable to receive size of the leak trace relative to the main trace.

potential A variable to receive the leak holding potential.

alternate A variable to receive the leak alternate protocol.

alternate_averaging A variable to receive the leak alternate averaging protocol.

delay A variable to receive the delay between the leak and the stimulus.

status A variable to receive the status of the operation.

Example

Variable status

Variable count

Variable scaling

Variable potential

Variable alternate

Variable alternate_averaging

Variable delay

PMSeriesGetLeak count, scaling, potential,
alternate, alternate_averaging, delay, status

Discussion

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

If alternate is non-zero every second leak pulse is inverted relative to the leak holding potential.

If alternate averaging is non-zero every second leak template is inverted relative to the leak holding potential when

averaging over multiple traces.

The delay is the time between the leak pulses and the main stimulus pulse. When delay is positive, the leak pulses follow the main stimulus, otherwise the leak pulses precede the main stimulus.

6.4.41 PMSeriesGetMarked

PMSeriesGetMarked returns whether the current series has been marked.

Syntax

```
PMSeriesGetMarked marked, status
```

marked An Long variable to receive the marked state.

status A variable to receive the status of the operation.

Example

```
Variable status  
Variable marked
```

```
PMSeriesGetMarked marked, status
```

Discussion

The current series must be set. To set the current series, use `PMSeriesSet`. [↗](#) *PMSeriesSet*, p. 186.

6.4.42 PMSeriesGetSampleInterval

PMSeriesGetSampleInterval returns the sampling interval in seconds for the current series.

Syntax

PMSeriesGetSampleInterval *interval, status*

interval A variable to receive the sampling interval.

status A variable to receive the status of the operation.

Example

Variable status
Variable interval

PMSeriesGetSampleInterval interval, status

Discussion

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

All channels are sampled at the same interval. The interval measures the time between samples on the same channel. However a given channel may be compressed. [↗ PMSeriesGetSampleInterval, p. 184.](#)

6.4.43 PMSeriesGetStart

PMSeriesGetStart returns the start time in seconds for the current series.

Syntax

PMSeriesGetStart *start, status*

start A variable to receive the start time.

status A variable to receive the status of the operation.

Example

Variable status
Variable series_start

PMSeriesGetStart series_start, status

Discussion

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

The start time of a series is measured in seconds since the beginning of the experiment.

6.4.44 PMSeriesGetTime

PMSeriesGetTime returns the date/time stamp for the start of the current series.

Syntax

PMSeriesGetTime *series_time, status*

series_time A variable to receive the time stamp.

status A variable to receive the status of the operation.

Example

Variable status
Variable series_time

PMSeriesGetTime series_time, status

Discussion

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

The time stamp is a numeric representation of the date and time for the series. It can be displayed in a table by setting the column format to be “data and time”.

6.4.45 PMSeriesGetUserName

PMSeriesGetUserName returns the user name supplied for the current series.

Syntax

PMSeriesGetUserName *name, status*

name A String variable to receive the user name

status A variable to receive the status of the operation.

Example

Variable status
String name

PMSeriesGetUserName name, status

Discussion

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

6.4.46 PMSeriesGetUserParameter

PMSeriesGetUserParameter returns the first user parameter for the current series.

Syntax

```
PMSeriesGetUserParameter index, name, value,
                        units, status
```

index A value specifying which parameter to return.

name A String variable to receive the **parameter name**.

value A variable to receive the parameter value.

units A String variable to receive the parameter units.

status A variable to receive the status of the operation.

Example

```
Variable status
String name
Variable parameter
String units
```

```
PMSeriesGetUserParameter 0, name,
parameter, units, status
```

Discussion

The index can range between 0 and 3.

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

6.4.47 PMSeriesSet

PMSeriesSet selects the current series.

Syntax

```
PMSeriesSet series, status
```

series The number of the series to use within the current group.

status A variable to receive the status of the operation.

Example

The following example sets the current series to the last series in the current group.

```
Variable status
Variable series
```

```
PMSeriesGetCount series, status
```

```
if (series < 1)
    | Handle the error
endif
```

```
PMSeriesSet series, status
```

Discussion

Series are numbered 1 to N, where N is the number of series in the current group. The number of series in the current group is returned by PMSeriesGetCount. [↗ PMSeriesGetCount, p. 181.](#)

6.4.48 PMStimulusGetCount

PMStimulusGetCount returns the number of stimulus channels used for the current series.

Syntax

PMStimulusGetCount *count, status*

count A Long variable to receive the number of stimulus channels used in the current series.

status A variable to receive the status of the operation.

Example

Variable status

Variable count

PMStimulusGetCount count, status

Discussion

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

6.4.49 PMStimulusGetDAC

PMStimulusGetDAC obtains the physical D/A channel.

Syntax

PMStimulusGetDAC *channel, dac, status*

channel A value representing the stimulus channel to use.

dac A variable to receive the D/A channel number.

status A variable to receive the status of the operation.

Example

Variable status

Variable dac

PMStimulusGetDAC 0, dac, status

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PMStimulusGetCount. [↗ PMStimulusGetCount, p. 187.](#)

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

The physical channel number is the channel label from the data acquisition device. The channel number is one of the values shown in table 96.

Table 96 Stimulus DAC

DAC	Output channel
0	DAC 0
1	DAC 1
2	DAC 2
3	DAC 3
5	Default

If the dac value is 'Default', then the channel is the default 'V-membrane Out' channel.

6.4.50 PMStimulusGetRange

PMStimulusGetRange returns the range of the stimulus.

Syntax

```
PMStimulusGetRange minimum, maximum,
status
```

minimum A variable to receive the lower limit.

maximum A variable to receive the upper limit.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable minimum
Variable maximum
```

```
PMStimulusGetRange minimum, maximum, status
```

Discussion

You must have a data set open.

PMStimulusGetRange returns the minimum and maximum possible values for the stimulus.

6.4.51 PMStimulusGetRelevantSegments

PMStimulusGetRelevantSegments returns the relevant x and y segments for the specified stimulus channel.

Syntax

```
PMStimulusGetRelevantSegments channel, x_segment,
y_segment, status
```

channel A value representing the stimulus channel to use.

x_segment A variable to receive the relevant x segment.

y_segment A variable to receive the relevant y segment.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable relevant_x
Variable relevant_y
```

```
PMStimulusGetRelevantSegments channel,
relevant_x, relevant_y, status
```

Discussion

The relevant x segment is the stimulus segment containing the relevant independent variable. The relevant y segment is the acquired data segment containing the “interesting” data.

The returned segment numbers will be in the range 1 to N, where N is the value returned by PMSegmentGetCount. [☞ PMSegmentGetCount, p. 179.](#)

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PMStimulusGetCount. [☞ PMStimulusGetCount, p. 187.](#)

The current series must be set. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

6.4.52 PMStimulusGetSource

PMStimulusGetSource returns the stimulus source for the specified stimulus channel.

Syntax

PMStimulusGetSource *channel, source, status*

channel A value representing the stimulus channel to use.

source A variable to receive the stimulus source.

status A variable to receive the status of the operation.

Example

Variable status
Variable source

```
PMStimulusGetSource 0, source, status
```

Discussion

The stimulus source is one of the values shown in table 98.

Table 97 Stimulus source

Source	Description
0	Template
1	File

If the source is 0, the stimulus was generated according to the stimulus protocol. If the source is 1, the stimulus came from a separate binary file containing an arbitrary waveform.

The current series must be set. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

6.4.53 PMStimulusGetStartSegment

PMStimulusGetStartSegment returns the starting segment and sample for the input channel tied to the specified stimulus channel.

Syntax

PMStimulusGetStartSegment *channel, segment, sample, status*

channel A value representing the stimulus channel to use.

segment A variable to receive the starting segment.

sample A variable to receive the start sample.

status A variable to receive the status of the operation.

Example

Variable status
Variable segment
Variable sample

```
PMStimulusGetStartSegment channel, segment,  
sample, status
```

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PMStimulusGetCount. [☞ PMStimulusGetCount, p. 187.](#)

The current series must be set. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

6.4.54 PMStimulusGetTriggerType

PMStimulusGetTriggerType returns the trigger type for the current series.

Syntax

PMStimulusGetTriggerType *type, status*

type A variable to receive the trigger type.

status A variable to receive the status of the operation.

Example

Variable status
Variable type

PMStimulusGetTriggerType type, status

Discussion

The trigger type is one of the values shown in table 98.

Table 98 Trigger types

type	trigger
0	None
1	Series
2	Sweep
3	Sweep no leak

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

6.4.55 PMStimulusGetUnits

PMStimulusGetUnits returns the units associated with the stimulus.

Syntax

PMStimulusGetUnits *channel, units, status*

channel A value representing the stimulus channel to use.

units A String variable to contain the units.

status A variable to receive the status of the operation.

Example

Variable status
String units

PMStimulusGetUnits 0, units, status

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PMStimulusGetCount. [↗ PMStimulusGetCount, p. 187.](#)

The current series must be set. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

6.4.56 PMSweepGetCount

PMSweepGetCount returns the number of sweeps in the current series.

Syntax

PMSweepGetCount *count, status*

count A variable to receive the number of sweeps recorded in the current series.

status A variable to receive the status of the operation.

Example

Variable status
Variable sweeps

PMSweepGetCount sweeps, status

Discussion

You must have a series selected. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in the series.

6.4.57 PMSweepGetDigitalIn

PMSweepGetDigitalIn the digital input value for the current sweep.

Syntax

PMSweepGetDigitalIn *value, status*

value A value to receive the digital value.

status A variable to receive the status of the operation.

Example

Variable status
Variable value

PMSweepGetDigitalIn value, status

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.58 PMSweepGetHoldingPotential

PMSweepGetHoldingPotential returns the holding potential for the current sweep.

Syntax

```
PMSweepGetHoldingPotential channel, potential,
status
```

channel A value representing the stimulus channel to use.

potential A variable to the sweep holding potential.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable potential
```

```
PMSweepGetHoldingPotential 0, potential,
status
```

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PMStimulusGetCount. [↗ PMStimulusGetCount, p. 187.](#)

6.4.59 PMSweepGetLabel

PMSweepGetLabel returns the label for the current sweep.

Syntax

```
PMSweepGetLabel label, status
```

label A String variable to receive the sweep label.

status A variable to receive the status of the operation.

Example

```
Variable status
String sweep_label
```

```
PMSweepGetLabel sweep_label, status
```

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.60 PMSweepGetMarked

PMSweepGetMarked returns whether the current sweep has been marked.

Syntax

PMSweepGetMarked *marked, status*

marked A variable to receive the marked state.

status A variable to receive the status of the operation.

Example

Variable status
Variable marked

PMSweepGetMarked marked, status

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [☞ PMSweepSet, p. 199.](#)

6.4.61 PMSweepGetRecordedCount

PMSweepGetRecordedCount returns the number of sweeps originally recorded for the current series.

Syntax

PMSweepGetRecordedCount *count, status*

count A variable to receive the number of sweeps recorded in the current series.

status A variable to receive the status of the operation.

Example

Variable status
Variable sweeps

PMSweepGetRecordedCount sweeps, status

Discussion

You must have a series selected. To set the current series, use PMSeriesSet. [☞ PMSeriesSet, p. 186.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in the series.

This count may be greater than the number of sweeps currently available if the data set has been edited. [☞ PMSweepGetCount, p. 191.](#)

6.4.62 PMSweepGetSampleCount

PMSweepGetSampleCount returns the number of samples in the current sweep.

Syntax

PMSweepGetSampleCount *channel, samples, status*

channel A value representing the channel to use.

samples A variable to receive the number of samples recorded in the current sweep.

status A variable to receive the status of the operation.

Example

Variable status
Variable samples

PMSweepGetSampleCount 0, samples, status

Discussion

The number of samples returned by PMSweepGetSampleCount is the number of samples recorded for the specified channel.

The sample count for different channels may differ if they are tied to different stimulus channels.

The number of samples recorded may be less than the total duration of all stimulus segments if recording does not begin at the beginning of the first stimulus segment. [☞ PMStimulusGetStartSegment, p. 189.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. To determine the number of channels, use `PMChannelGetCount`. [☞ PMChannelGetCount, p. 165.](#)

You must have a sweep selected. To set the current sweep, use `PMSweepSet`. [☞ PMSweepSet, p. 199.](#)

6.4.63 PMSweepGetSolutions

PMSweepGetSolutions returns the solution numbers for the current sweep.

Syntax

PMSweepGetSolutions *internal, external, status*

internal A variable to receive the internal solution number.

external A variable to receive the external solution number.

status A variable to receive the status of the operation.

Example

Variable status
Variable internal
Variable external

PMSweepGetSolutions internal, external, status

Discussion

You must have a sweep selected. To set the current sweep, use `PMSweepSet`. [☞ PMSweepSet, p. 199.](#)

6.4.64 PMSweepGetStart

PMSweepGetStart returns the sweep start time.

Syntax

PMSweepGetStart *sweep_time, data_time, status*

sweep_time A variable to receive the starting time of the sweep.

data_time A variable to receive the starting time of the sweep data.

status A variable to receive the status of the operation.

Example

Variable status
Variable sweep_time
Variable data_time

PMSweepGetStart sweep_time, data_time, status

Discussion

The sweep time is measured in seconds since the beginning of the series. This is the time the stimulus starts.

The data time is measured in seconds since the beginning of the series. This is the starting time of recorded data.
☞ *PMStimulusGetStartSegment*, p. 189.

You must have a sweep selected. To set the current sweep, use PMSweepSet. ☞ *PMSweepSet*, p. 199.

6.4.65 PMSweepGetTemperature

PMSweepGetTemperature returns the temperature for the current series.

Syntax

PMSweepGetTemperature *temperature, status*

temperature A Double variable to receive the **temperature**.

status A variable to receive the status of the operation.

Example

Variable status
Variable temperature

PMSweepGetTemperature temperature, status

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. ☞ *PMSweepSet*, p. 199.

6.4.66 PMSweepGetTimer

PMSweepGetTimer returns the timer value at the start of the sweep.

Syntax

PMSweepGetTimer *timer, status*

timer A variable to receive the timer value at the start of the sweep.

status A variable to receive the status of the operation.

Example

Variable timer
Variable status

PMSweepGetTimertimer, status

Discussion

The timer value is measured in seconds since the most recent timer reset.

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

6.4.67 PMSweepGetUserParameter

PMSweepGetUserParameter returns the first user parameter for the current series.

Syntax

PMSweepGetUserParameter *index, name, value, units*

index A value specifying which parameter to return.

name A String variable to receive the parameter name.

value A Double variable to receive the parameter value.

units A String variable to receive the parameter units.

status A variable to receive the status of the operation.

Example

String name As Variant
Variable parameter As Double
String units As Variant

PMSweepGetUserParameter 0, name,
parameter, units

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

The index can range between 0 and 3.

6.4.68 PMSweepRead

PMSweepRead reads the data for the specified channel of the current sweep into a wave.

Syntax

PMSweepRead *channel, start, samples, data, status*

channel The channel number containing the data to read.

start The number of the first sample within the sweep to read.

samples The number of samples to read.

data A wave to receive the data. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status

Variable samples

PMSweepGetSampleCount 0, samples, status

Make/D/N=(samples) data

PMSweepRead channel, 0, samples, data, status

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [↗ PMChannelGetCount, p. 165.](#)

The number of samples in the current sweep is provided by PMSweepGetSampleCount. [↗ PMSweepGetSampleCount, p. 194.](#)

6.4.69 PMSweepReadLeak

PMSweepReadLeak reads the leak template for the specified channel of the current sweep into a wave.

Syntax

PMSweepReadLeak *channel, start, samples, data, status*

channel The channel number containing the data to read.

start The number of the first sample within the sweep to read.

samples The number of samples to read.

data A wave to receive the data. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status

Variable samples

PMSweepGetSampleCount channel, samples, status

Make/D/N=(samples) leak_data

PMSweepReadLeak 0, samples, leak_data, status

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

The number of samples in the current sweep is provided by PMSweepGetSampleCount. [↗ PMSweepGetSampleCount, p. 194.](#)

You can use PMChannelGetLeakAvailable to determine if leak data is available for the current sweep. [↗ PMChannelGetLeakAvailable, p. 166.](#)

6.4.70 PMSweepReadRaw

PMSweepReadRaw reads raw data without leak correction for the specified channel of the current sweep into a wave.

Syntax

PMSweepReadRaw *channel, start, samples, data, status*

channel The channel number containing the data to read.

start The number of the first sample within the sweep to read.

samples The number of samples to read.

data A wave to receive the data. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status
Variable samples

PMSweepGetSampleCount channel, samples,
status

Make/D/N=(samples) data

PMSweepReadRaw channel, 0, samples, data,
status

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [↗ PMChannelGetCount, p. 165.](#)

The number of samples in the current sweep is provided by PMSweepGetSampleCount. [↗ PMSweepGetSampleCount, p. 194.](#)

You can use PMChannelGetLeakAvailable to determine if leak data is available for the current sweep. [↗ PMChannelGetLeakAvailable, p. 166.](#)

6.4.71 PMSweepReadStimulus

PMSweepReadStimulus reads the output data for the specified stimulus channel of current sweep into a wave.

Syntax

PMSweepReadStimulus *channel, start, samples, stimulus, status*

channel The stimulus channel to use.

start The number of the first sample within the sweep to read.

samples The number of samples to read.

stimulus A wave to receive the stimulus. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status
Variable samples

PMSweepGetSampleCount 0, samples, status

Make/D/N=(samples) data

PMSweepReadStimulus 0, 0, samples, data, status

Discussion

You must have a sweep selected. To set the current sweep, use PMSweepSet. [↗ PMSweepSet, p. 199.](#)

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PMStimulusGetCount. [↗ PMStimulusGetCount, p. 187.](#)

The number of samples in the current sweep is provided by PMSweepGetSampleCount. [↗ PMSweepGetSampleCount, p. 194.](#)

6.4.72 PMSweepSet

PMSweepSet selects the current sweep.

Syntax

```
PMSweepSet sweep, status
```

sweep A value representing the sweep to set.

status A variable to receive the status of the operation.

Example

This example sets the current sweep to the last sweep in the current series.

```
Variable status
```

```
Variable sweeps
```

```
PMSweepGetCount sweeps, status
```

```
if (sweeps < 1)
```

```
  | Handle the error
```

```
endif
```

```
PMSweepSet sweeps, status
```

Discussion

Sweeps are numbered 1 to N, where N is the number of sweeps in the current series. The number of sweeps in the current series is provided by PMSweepGetCount. [↗ PMSweepGetCount, p. 191.](#)

You must have a series selected. To set the current series, use PMSeriesSet. [↗ PMSeriesSet, p. 186.](#)

7 PatchMaster: Visual Basic

Section	
7.1 Using DataAccess	P. 200
7.2 Getting Started	P. 200
7.3 Robust Processing	P. 203
7.4 Operations	P. 204
7.5 Reference	P. 206

7.1 Using DataAccess

To make DataAccess available to a Visual Basic program, add the file PatchMasterVB.bas to the project. Ensure that the file PatchMasterVB.dll is in your path.

To make DataAccess available to an Systat Software SigmaPlot script module, include the following statement in the module:

```
'#Uses "PatchMasterVB.bas"
```

The leading quote (!) is required. The bas file must be in the same folder as your SigmaPlot project, or you must explicitly specify the path to the bas file.

7.1.1 Status

Each operation returns a status code. This code is zero if no error is detected.

If an error is detected, the returned status value is non-zero. It can be translated to a text string using PatchMasterGet-StatusText.

7.1.2 Data Types

Integer values are passed as the data type “Long”. Real values are passed as the data type “Double”. Arrays are passed by passing the first element in the array. Character string values are passed as the data type “Variant”. The data type “String” is not used because when Visual Basic passes parameters to dlls, it performs undesired translations from Unicode to ASCII.

7.1.3 Values

All duration values are measured in seconds.

All amplitude values are measured in volts, taking into account any scaling specified in the file.

7.1.4 Calls

Almost all DataAccess procedures return a value. In Visual Basic, you have several choices of syntax when you call such procedures. For example, you can call PatchMasterChannelGetCount as follows:

- 1 As a function, using the return value:

```
result = PatchMasterChannelGetCount(session, count)
```

- 2 As a subroutine using “Call”:

```
Call PatchMasterChannelGetCount(session, count)
```

- 3 Directly as a command. In this case, the parameters are not enclosed in parentheses:

```
PatchMasterChannelGetCount session, count
```

Any of the methods work. The examples in this manual generally use the command form.

7.2 Getting Started

This section explains how to access an PatchMaster data

set. It contains a step by step description of the operations to perform to open a data set and read the data in it.

7.2.1 Opening a Session

Open a session using the PatchMasterSessionOpen operation. [☞ PatchMasterSessionOpen, p. 232.](#)

PatchMasterSessionOpen returns a handle. You must pass this handle to other operations that use that session.

7.2.2 Opening a Data Set

Open an PatchMaster data set using the PatchMasterFileOpen operation. [☞ PatchMasterFileOpen, p. 219.](#)

In order to open a data set, you must supply the path to the DAT file.

The following example opens a session, then opens a data set with the path “f:\test.dat”. After the data set is open, it closes the data set and the session.

```
Dim session As Long
Dim status As Long
Dim file_name As Variant
file_name = "f:\test.dat"

PatchMasterSessionOpen session
status = PatchMasterFileOpen(session, file_name)

If status <> 0 Then
    ' The data set could not be opened. Handle the
    ' error and do not call PatchMasterFileClose
End If

PatchMasterFileClose session
PatchMasterSessionClose session
```

7.2.3 File Parameters

Once you have opened a data set, you can determine the parameters of the data set. The parameters include the creation date and time and the data set comment. [☞ File Parameter Operations, p. 204.](#)

The following example obtains data set creation date and

time:

```
Dim file_date_time As Date

PatchMasterFileGetTime session, file_date_time
```

7.2.4 Groups

A data set is a sequence of groups. You use the group operations to select a group and determine its parameters. [☞ Group Operations, p. 204.](#)

You use PatchMasterGroupGetCount to determine the number of groups in a data set. [☞ PatchMasterGroupGetCount, p. 221.](#)

Groups are numbered 1 to N, where N is the number of groups in a data set. You use PatchMasterGroupSet to set the group to use. [☞ PatchMasterGroupSet, p. 223.](#)

The following example obtains the label of each group in a data set. The label is a text string.

```
Dim group_count As Long
Dim current_group As Long
Dim group_label As Variant

PatchMasterGroupGetCount session, group_count

If group_count > 0 Then
    For current_group = 1 To group_count
        PatchMasterGroupSet session, current_group
        PatchMasterGroupGetLabel session, group_label
        ' Process the label here
    Next current_group
Else
    ' The data set contains no groups
End If
```

A file may have no groups if no data was acquired.

7.2.5 Series

A group is a sequence of series. You use the series operations to select a series and determine its parameters. [☞ Series Operations, p. 205.](#)

You use PatchMasterSeriesGetCount to determine the number of series in a group. [☞ PatchMasterSeriesGetCount, p. 226.](#)

Series are numbered 1 to N, where N is the number of series in a group. You use PatchMasterSeriesSet to set the series to use. [☞ PatchMasterSeriesSet, p. 231.](#)

The following example obtains the label of each series in the current group. The label is a text string.

```
Dim series_count As Long
Dim current_series As Long
Dim series_label As Variant

PatchMasterSeriesGetCount session, series_count

If series_count > 0 Then
  For current_series = 1 To series_count
    PatchMasterSeriesSet session, current_series
    PatchMasterSeriesGetLabel session, series_label
    ' Process label here
  Next current_series
Else
  ' The group contains no series
End If
```

A group may have no series if no data was acquired.

7.2.6 Sweeps

A series is a sequence of sweeps. You use the sweep operations to select a sweep and determine its parameters. [☞ Sweep Operations, p. 205.](#)

You use PatchMasterSweepGetCount to determine the number of sweeps in a sweep. [☞ PatchMasterSweepGetCount, p. 237.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in a series. You use PatchMasterSweepSet to set the sweep to use. [☞ PatchMasterSweepSet, p. 246.](#)

The following example obtains the number of samples in each sweep in the current series.

```
Dim sweep_count As Long
Dim current_sweep As Long
Dim sweep_samples As Long

PatchMasterSweepGetCount session, sweep_count

If sweep_count > 0 Then
  For current_sweep = 1 To sweep_count
    PatchMasterSweepSet session, current_sweep
```

```
PatchMasterSweepGetSampleCount
  session, 0, sweep_samples
  ' Process sample count here
Next current_sweep
Else
  ' The series contains no sweep
End If
```

A series may have no sweep if no data was acquired.

7.2.7 Segments

A stimulus template is described by a sequence of segments. Although all sweeps in a series use the same stimulus template, the duration and voltage of each segment can vary with each sweep.

Segment numbers range from 1 to N, where N is the number of segments in the series. You use PatchMasterSegmentGetCount to determine the number of segments in a series. [☞ PatchMasterSegmentGetCount, p. 224.](#)

7.2.8 Channels

PatchMasterChannelGetCount returns the number of channels recorded in a sweep. Normally the number of channels recorded in each sweep in a series will be the same when a data set is acquired, but channels can be removed from individual sweeps during data set editing, leaving a series with varying numbers of channels recorded in each sweep. [☞ PatchMasterChannelGetCount, p. 210.](#)

The following example obtains the number of channels recorded in the current sweep:

```
Dim channel_count As Long

PatchMasterChannelGetCount session, channel_count
```

Channel numbers range from 0 to N-1, where N is the number of channels in the sweep. Existing versions of PatchMaster never record more than two channels in a sweep.

7.2.9 Reading Data

To read a sweep, use PatchMasterSweepRead. PatchMasterSweepRead can read either a whole sweep or sections of a sweep. [☞ PatchMasterSweepRead, p. 243.](#)

Normally one would read whole sweeps but it may be necessary to read sections of a sweep if you have insufficient memory available to read whole sweeps.

Reading Sweeps

The following example reads each sweep of a series into an array.

```
Dim sweep_count As Long
Dim current_sweep As Long
Dim samples As Long
Dim buffer() As Double

PatchMasterSweepGetCount session, segment,
sweep_count

If sweep_count > 0 Then
  For current_sweep = 1 To sweep_count
    PatchMasterSweepSet session, current_sweep
    PatchMasterSweepGetSampleCount session, 0,
      samples
    ReDim buffer(samples-1)

    PatchMasterSweepRead session, channel, 0,
samples,
  buffer(0)
  ' Data processing goes here
  Next current_sweep
Else
  ' The series has no sweeps
End If
```

Reading Sections of a Sweep

The following example reads the first “length” values of the first sweep into an array.

```
Dim samples As Long
Dim buffer() As Double

PatchMasterSweepGetSampleCount session, channel,
samples

If samples > length Then
  samples = length
End If
```

```
ReDim buffer(samples-1)
```

```
PatchMasterSweepRead session, channel, 0, samples,
buffer(0)
```

7.2.10 Closing a Data Set

To close a data set, use PatchMasterFileClose. [PatchMasterFileClose](#), p. 216.

The following example closes a PatchMaster data set.

```
PatchMasterFileClose session
```

7.2.11 Closing a Session

To close a session, use PatchMasterSessionClose. [PatchMasterSessionClose](#), p. 232.

7.3 Robust Processing

If you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

All operations return a status value of type Long. The value is zero if the operation was successful, and non-zero if an error was detected.

If an error is detected, you can use PatchMasterGetStatusText to translate the status code to a message. [PatchMasterGetStatusText](#), p. 220.

The following example shows how the user might be allowed to select a file. The example does not terminate until the user selects a valid PatchMaster data set.

```
Dim filespec As Variant
Dim status As Long
Dim session As Long

PatchMasterSessionOpen session

Do
  ' Add code here to obtain the file specification in "filespec"
  status = PatchMasterFileOpen(session, filespec)

  If status = 0 Then
```

```

Exit Do
End If

' Translate the status code to a message here and
' display it for the user
Loop

```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

7.4 Operations

This chapter describes each of the operations provided by DataAccess. The operations are grouped by category.

7.4.1 Common Operations

The common operations are valid regardless of whether a data set is open or not. The common operations are shown in table 99.

Table 99 Common operations

Operation	
PatchMasterGetVersion	p. 220
PatchMasterGetStatusText	p. 220

7.4.2 Session Operations

The session operations allow you to open and close a session. The session operations are shown in

Table 100 Session operations

Operation	
PatchMasterSessionOpen	p. 232
PatchMasterSessionClose	p. 232

7.4.3 File Operations

The file operations allow you to access a PatchMaster data

set. The file operations are shown in table 101.

Table 101 File operations

Operation	
PatchMasterFileGetPatchMasterVersion	p. 218
PatchMasterFileClose	p. 216

The file operations are valid when a session is open.

7.4.4 File Parameter Operations

The file parameter operations allow you to obtain data set parameters. The file parameter operations are shown in table 102.

Table 102 File parameter operations

Operation	
PatchMasterFileGetComment	p. 216
PatchMasterFileGetMarked	p. 217
PatchMasterFileGetTime	p. 218
PatchMasterFileGetSignature	p. 217
PatchMasterFileGetPatchMasterVersion	p. 218
PatchMasterFileGetVersion	p. 219

The file parameter operations are valid when a data set is open.

7.4.5 Group Operations

The group operations allow you to access each group of a data set. The group operations are shown in table 103.

Table 103 Group operations

Operation	
PatchMasterGroupGetComment	p. 221
PatchMasterGroupGetCount	p. 221
PatchMasterGroupGetExperiment	p. 222
PatchMasterGroupGetLabel	p. 222
PatchMasterGroupGetMarked	p. 223
PatchMasterGroupSet	p. 223

The group operations are valid when a data set is open. Most require that a current group be set.

7.4.6 Series Operations

The series operations allow you to access each series of a group. The series operations are shown in table 104.

Table 104 Series operations

Operation	
PatchMasterSeriesGetComment	p. 226
PatchMasterSeriesGetCount	p. 226
PatchMasterSeriesGetLabel	p. 227
PatchMasterSeriesGetLeak	p. 227
PatchMasterSeriesGetMarked	p. 228
PatchMasterSeriesGetSampleInterval	p. 229
PatchMasterSeriesGetStart	p. 229
PatchMasterSeriesGetTime	p. 230
PatchMasterSeriesGetUserParameter	p. 231
PatchMasterSeriesSet	p. 231

The series operations are valid when a data set is open. Most require that a current series be set.

7.4.7 Sweep Operations

The sweep operations allow you to access each sweep of a series. The sweep operations are shown in table 105.

Table 105 Sweep operations

Operation	
PatchMasterSweepGetCount	p. 237
PatchMasterSweepGetDigitalIn	p. 237
PatchMasterSweepGetHoldingPotential	p. 238
PatchMasterSweepGetLabel	p. 238
PatchMasterSweepGetMarked	p. 239
PatchMasterSweepGetRecordedCount	p. 239
PatchMasterSweepGetSampleCount	p. 240
PatchMasterSweepGetSolutions	p. 240
PatchMasterSweepGetStart	p. 241
PatchMasterSweepGetTemperature	p. 241
PatchMasterSweepGetTimer	p. 242
PatchMasterSweepGetUserParameter	p. 242
PatchMasterSweepRead	p. 243
PatchMasterSweepReadLeak	p. 244
PatchMasterSweepReadRaw	p. 245
PatchMasterSweepReadStimulus	p. 245
PatchMasterSweepSet	p. 246

The sweep operations are valid when a data set is open. Most require that a current sweep be set.

7.4.8 Channel Operations

The channel operations allow you to obtain information about the acquired data channels. The channel operations are shown in table 106.

Table 106 Channel operations

Operation	
PatchMasterChannelGetADC	p. 206
PatchMasterChannelGetAmplifier	p. 207
PatchMasterChannelGetAvailable	p. 207
PatchMasterChannelGetBandwidth	p. 208
PatchMasterChannelGetCellPotential	p. 208
PatchMasterChannelGetCompression	p. 209
PatchMasterChannelGetCount	p. 210
PatchMasterChannelGetLabel	p. 210
PatchMasterChannelGetLeakAvailable	p. 211
PatchMasterChannelGetLinkedStimulus	p. 211
PatchMasterChannelGetMarked	p. 212
PatchMasterChannelGetPipetteResistance	p. 212
PatchMasterChannelGetRange	p. 213
PatchMasterChannelGetSampleInterval	p. 214
PatchMasterChannelGetSealResistance	p. 214
PatchMasterChannelGetStart	p. 215
PatchMasterChannelGetUnits	p. 215

7.4.9 Stimulus Operations

The stimulus operations allow you to obtain information about the stimulus protocol. The stimulus operations are shown in table 107.

Table 107 Stimulus operations

Operation	
PatchMasterStimulusGetCount	p. 233
PatchMasterStimulusGetDAC	p. 233
PatchMasterStimulusGetRange	p. 234
PatchMasterStimulusGetRelevantSegments	p. 234
PatchMasterStimulusGetSource	p. 235
PatchMasterStimulusGetStartSegment	p. 235
PatchMasterStimulusGetStartSegment	p. 235
PatchMasterStimulusGetUnits	p. 236

7.4.10 Segment Operations

The segment operations allow you to access information about the stimulus segments. The segment operations are shown in table 108.

Table 108 Segment operations

Operation	
PatchMasterSegmentGetCount	p. 224
PatchMasterSegmentGetParameters	p. 224

7.5 Reference

This section lists each DataAccess operation in alphabetical order.

7.5.1 PatchMasterChannelGetADC

PatchMasterChannelGetADC obtains the physical A/D channel.

Syntax

PatchMasterChannelGetADC *session, channel, adc*

session A Long value representing the current session.

channel A Long value representing the channel.

adc An Long variable to receive the A/D channel number.

Example

```
Dim adc As Long
```

```
PatchMasterChannelGetADC session, channel, adc
```

Discussion

The logical channel number may differ from the physical A/D channel used. The physical channel number is the one labeled on the data acquisition device.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.2 PatchMasterChannelGetAmplifier

PatchMasterChannelGetAmplifier returns the amplifier settings for the current sweep for the specified channel.

Syntax

PatchMasterChannelGetAmplifier *session, channel, adc*

session A Long value representing the current session.

channel A Long value representing the channel.

gain A Double value to receive the gain.

capacitance A Double value to receive the capacitance.

conductance A Double value to receive the series conductance.

resistance A Double value to receive the series resistance compensation.

Example

```
Dim gain As Double
Dim capacitance As Double
Dim conductance As Double
Dim resistance As Double
```

```
PatchMasterChannelGetAmplifier session, channel,
    gain, capacitance, conductance, resistance
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.3 PatchMasterChannelGetAvailable

PatchMasterChannelGetAvailable returns whether the channel trace is available for the current sweep.

Syntax

PatchMasterChannelGetAvailable *session, channel, available*

session A Long value representing the current session.

channel A Long value representing the channel.

available An Long variable to receive the available status.

Example

```
Dim available As Long
```

```
PatchMasterChannelGetAvailable session, channel,
    available
```

```
If available <> 0 Then
    ' Process the trace
End If
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

It is possible to edit data files in PatchMaster, so a trace for a given sweep can be deleted from the data file.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.4 PatchMasterChannelGetBandwidth

PatchMasterChannelGetBandwidth returns the bandwidth for the specified channel for the current sweep.

Syntax

```
PatchMasterChannelGetBandwidth session, channel,
    bandwidth
```

session A Long value representing the current session.

channel A Long value representing the channel.

bandwidth A Double variable to receive the bandwidth.

Example

```
Dim bandwidth As Double
```

```
PatchMasterChannelGetBandwidth session, channel,
    bandwidth
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.5 PatchMasterChannelGetCellPotential

PatchMasterChannelGetCellPotential returns the cell potential for the specified channel for the current sweep.

Syntax

```
PatchMasterChannelGetCellPotential session, channel,
    cell_potential
```

session A Long value representing the current session.

channel A Long value representing the channel.

cell_potential A Double variable to receive the cell potential.

Example

```
Dim cell_potential As Double
```

```
PatchMasterChannelGetCellPotential session, channel,
    cell_potential
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.6 PatchMasterChannelGetCompression

PatchMasterChannelGetCompression returns the data compression applied to the specified channel.

Syntax

```
PatchMasterChannelGetCompression session, channel,  
factor, skip, averaged, filtered
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

factor A Long variable to receive the decimation factor.

skip A Long variable to receive the start offset.

filtered A Long variable to receive the filtered mode.

status A Long variable to receive the status of the operation.

Example

```
Dim factor As Long  
Dim skip As Long  
Dim averaged As Long  
Dim filtered As Long
```

```
PatchMasterChannelGetCompression session, 0, factor,  
skip, averaged, filtered
```

Discussion

PatchMasterChannelGetCompression returns the compression parameters for the specified channel. The data is decimated by the decimation factor. The decimation can start at an offset returned by the skip parameter. The offset will always be less than decimation factor.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

If averaged is non-zero, the samples are averaged together, otherwise single samples are used.

If filtered is non-zero, a digital filter has been applied to the data.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.7 PatchMasterChannelGetCount

PatchMasterChannelGetCount returns the number of channels recorded in the current series.

Syntax

```
PatchMasterChannelGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of channels recorded in the current series.

Example

```
Dim channels As Long
```

```
PatchMasterChannelGetCount session, channels
```

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [☞ PatchMasterSeriesSet, p. 231.](#)

7.5.8 PatchMasterChannelGetLabel

PatchMasterChannelGetLabel returns the label for the specified channel.

Syntax

```
PatchMasterChannelGetLabel session, channel, label
```

session A Long value representing the current session.

channel A Long value representing the channel.

label A Variant variable to receive the label.

Example

```
Dim label As Variant
```

```
PatchMasterChannelGetLabel session, channel, label
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [☞ PatchMasterChannelGetCount, p. 210.](#)

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [☞ PatchMasterSweepSet, p. 246.](#)

7.5.9 PatchMasterChannelGetLeakAvailable

PatchMasterChannelGetLeakAvailable returns whether the channel's leak trace is available for the current sweep.

Syntax

```
PatchMasterChannelGetLeakAvailable session, channel,
    available
```

session A Long value representing the current session.

channel A Long value representing the channel.

available An Long variable to receive the available status.

Example

```
Dim available As Long
```

```
PatchMasterChannelGetLeakAvailable session, channel,
    available
```

```
If available <> 0 Then
    ' Process the leak trace
End If
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

It is possible to edit data files in PatchMaster, so a leak trace for a given sweep can be deleted from the data file.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.10 PatchMasterChannelGetLinkedStimulus

PatchMasterChannelGetLinkedStimulus returns the stimulus channel associated with the specified channel.

Syntax

```
PatchMasterChannelGetLinkedStimulus session, channel,
    linked_channel
```

session A Long value representing the current session.

channel A Long value representing the channel.

linked_channel An Long variable to receive the linked stimulus channel number.

Example

```
Dim linked_channel As Long
```

```
PatchMasterChannelGetLinkedStimulus session,
    channel, linked_channel
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.11 PatchMasterChannelGetMarked

PatchMasterChannelGetMarked returns whether the specified channel has been marked.

Syntax

PatchMasterChannelGetMarked *session, channel, marked*

session A Long value representing the current session.

channel A Long value representing the channel.

marked An Long variable to receive the marked state.

Example

Dim marked As Long

PatchMasterChannelGetMarked session, channel,
marked

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.12 PatchMasterChannelGetPipetteResistance

PatchMasterChannelGetPipetteResistance returns the pipette resistance for the specified channel.

Syntax

PatchMasterChannelGetPipetteResistance *session, channel, resistance*

session A Long value representing the current session.

channel A Long value representing the channel to use.

resistance A Double variable to receive the pipette resistance.

Example

Dim count As Long

Dim resistance As Double

PatchMasterChannelGetPipetteResistance session,
channel, resistance

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.13 PatchMasterChannelGetRange

PatchMasterChannelGetRange returns the range of the specified channel.

Syntax

```
PatchMasterChannelGetRange session, channel, minimum,
    maximum
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

minimum A Double variable to receive the lower limit.

maximum A Double variable to receive the upper limit.

Example

```
Dim minimum As Double
Dim maximum As Double
```

```
PatchMasterChannelGetRange session, 0, minimum,
    maximum
```

Discussion

PatchMasterChannelGetRange returns the minimum and maximum possible values for the specified channel.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

7.5.14 PatchMasterChannelGetRecordingMode

PatchMasterChannelGetRecordingMode returns the recording mode for the specified channel.

Syntax

```
PatchMasterChannelGetRecordingMode session, channel,
    mode
```

session A Long value representing the current session.

channel A Long value representing the channel.

mode A Long variable to receive the recording mode.

Example

```
Dim mode As Long
```

```
PatchMasterChannelGetRecordingMode session,
    channel, mode
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

The recording mode is one of the values listed in table 109.

Table 109 Recording modes

Mode	Protocol
0	In out
1	On cell
2	Out out
3	Whole cell
4	C clamp
5	V clamp

7.5.15 PatchMasterChannelGetSampleInterval

PatchMasterChannelGetSampleInterval returns the sampling interval in seconds for the specified channel.

Syntax

```
PatchMasterChannelGetSampleInterval session, channel,
interval
```

session A Long value representing the current session.

channel A Long value representing the channel.

interval A Double variable to receive the sampling interval.

Example

```
Dim interval As Double
```

```
PatchMasterChannelGetSampleInterval session, channel,
interval
```

Discussion

All channels are sampled at the same interval. The interval measures the time between samples on the same channel. However a given channel may be compressed, resulting in a larger interval than for the series. [☞ PatchMasterSeriesGetSampleInterval, p. 229.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use `PatchMasterChannelGetCount`. [☞ PatchMasterChannelGetCount, p. 210.](#)

The current series must be set. To set the current series, use `PatchMasterSeriesSet`. [☞ PatchMasterSeriesSet, p. 231.](#)

7.5.16 PatchMasterChannelGetSealResistance

PatchMasterChannelGetSealResistance returns the seal resistance for the specified channel.

Syntax

```
PatchMasterChannelGetSealResistance session, channel,
resistance
```

session A Long value representing the current session.

channel A Long value representing the channel.

resistance A Double variable to receive the seal resistance.

Example

```
Dim resistance As Double
```

```
PatchMasterChannelGetSealResistance session, channel,
resistance
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use `PatchMasterChannelGetCount`. [☞ PatchMasterChannelGetCount, p. 210.](#)

The current sweep must be set. To set the current sweep, use `PatchMasterSweepSet`. [☞ PatchMasterSweepSet, p. 246.](#)

7.5.17 PatchMasterChannelGetStart

PatchMasterChannelGetStart returns the precise start time in seconds relative to the stimulus start for the specified channel.

Syntax

```
PatchMasterChannelGetStart session, channel,
    resistance
```

session A Long value representing the current session.

channel A Long value representing the channel.

time A Double variable to receive the start time.

Example

```
Dim time As Double
```

```
PatchMasterChannelGetStart session, channel,
    time
```

Discussion

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current sweep must be set. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

The start time may include offsets due to a nonzero start segment and time set for the stimulus protocol. It may also include sampling sequence delays.

7.5.18 PatchMasterChannelGetUnits

PatchMasterChannelGetUnits obtains the units associated with a channel.

Syntax

```
PatchMasterChannelGetUnits session, units
```

session A Long value representing the current session.

channel A Long value representing the channel.

units A Variant variable to receive the units.

Example

```
Dim units As Variant
```

```
PatchMasterChannelGetUnits session, channel, units
```

Discussion

The units are entered by the user.

Channels are numbered 0 to N-1, where N is the number of channels in the series. To determine the number of channels, use PatchMasterChannelGetCount. [PatchMasterChannelGetCount](#), p. 210.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.19 PatchMasterFileClose

PatchMasterFileClose terminates processing of the PatchMaster data set.

Syntax

```
PatchMasterFileClose session
```

session A Long value representing the current session.

Example

```
PatchMasterFileClose session
```

Discussion

You must have a data set open.

You should perform PatchMasterFileClose as soon as you complete processing of a data set.

7.5.20 PatchMasterFileGetComment

PatchMasterFileGetComment returns the comment for the data set.

Syntax

```
PatchMasterFileGetComment session, comment
```

session A Long value representing the current session.

comment A Variant variable in which the data set comment is returned.

Example

```
Dim text As Variant
```

```
PatchMasterFileGetComment session, text
```

Discussion

You must have a data set open.

7.5.21 PatchMasterFileGetMarked

PatchMasterFileGetMarked returns whether the file has been marked.

Syntax

PatchMasterFileGetMarked *session, channel, marked*

session A Long value representing the current session.

marked An Long variable to receive the marked state.

Example

```
Dim marked As Long
```

```
PatchMasterFileGetMarked session, channel,  
    marked
```

Discussion

You must have a data set open.

7.5.22 PatchMasterFileGetSignature

PatchMasterFileGetSignature returns a unique signature for the data file.

Syntax

PatchMasterFileGetSignature *session, signature*

session A Long value representing the current session.

signature A Variant variable in which the file signature is returned.

Example

```
Dim signature As Variant
```

```
PatchMasterFileGetSignature session, signature
```

Discussion

You must have a data set open.

The returned signature is a string containing a 32 character hexadecimal value.

The signature is calculated from information in the file. Except for a very unlikely coincidence, the signature should uniquely identify the file.

7.5.23 PatchMasterFileGetTime

PatchMasterFileGetTime returns the creation date and time of the data set.

Syntax

```
PatchMasterFileGetTime session, date_time
```

session A Long value representing the current session.

date_time A Date variable to receive the data set creation date and time.

Example

```
Dim date_time As Date
```

```
PatchMasterFileGetTime session, date_time
```

Discussion

You must have a data set open.

7.5.24 PatchMasterFileGetPatchMasterVersion

PatchMasterFileGetPatchMasterVersion returns the version of PatchMaster which created the file.

Syntax

```
PatchMasterFileGetPatchMasterVersion session, version
```

session A Long value representing the current session.

version A Long variable to receive the PatchMaster version.

Example

```
Dim version As Variant
```

```
PatchFileMasterGetPatchMasterVersion session, version
```

Discussion

You must have a data set open.

7.5.25 PatchMasterFileGetVersion

PatchMasterFileGetVersion returns the PatchMaster file format version of the file.

Syntax

PatchMasterFileGetVersion *session, version*

session A Long value representing the current session.

version A Variant variable to receive the file version.

Example

```
Dim version As Variant
```

```
PatchMasterFileGetVersion session, version
```

Discussion

You must have a data set open.

7.5.26 PatchMasterFileOpen

PatchMasterFileOpen prepares a PatchMaster data set for processing.

Syntax

PatchMasterFileOpen *session, path*

session A Long value representing the current session.

path A Variant value representing the data set path.

Example

```
Dim file_name As Variant
```

```
file_name = "Data.dat"
```

```
PatchMasterFileOpen session, file_name
```

Discussion

You must not have a data set open.

A PatchMaster data set consists of three files, with the extensions PUL, PGF, and DAT. All three files must be present in the same directory for PatchMasterFileOpen to succeed.

Once a data set is open, you can access data set parameters. [☞ File Parameter Operations, p. 204](#). You can process the groups in the data set. [☞ Group Operations, p. 204](#).

When you are done with a data set, you should close it. [☞ PatchMasterFileClose, p. 216](#).

PatchMasterFileOpen invalidates the current group. You must set the current group using PatchMasterGroupSet. [☞ PatchMasterGroupSet, p. 223](#).

7.5.27 PatchMasterGetStatusText

PatchMasterGetStatusText translates a status value returned by a DataAccess operation to a text string.

Syntax

```
PatchMasterGetStatusText session, status, message
```

session A Long value representing the current session.

status A Long value representing the status value to translate.

message A Variant variable in which the translated text is returned.

Example

```
Dim status As Long
Dim message As Variant
Dim file_name As Variant
file_name = "Sample.dat"

status = PatchMasterFileOpen(session, file_name)

If status <> 0 Then
    PatchMasterGetStatusText session, status, message
    ' Display the message here
End If
```

Discussion

PatchMasterGetStatusText does not return a value, so it should not be called as a function.

You need not have a data set open.

PatchMasterGetStatusText is the only means provided to interpret status values.

7.5.28 PatchMasterGetVersion

PatchMasterGetVersion returns the version number of the DataAccess package.

Syntax

```
PatchMasterGetVersion version
```

version A Long variable to receive the DataAccess version number.

Example

```
Dim version As Long

PatchMasterGetVersion version
```

Discussion

You need not have a data set open.

If you are writing a set of procedures using DataAccess, you can use PatchMasterGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess. You can use the following code to ensure that this version or a later version is in use:

```
Dim version As Long
PatchMasterGetVersion version

If version < N Then
    ' Handle the error
End If
```

7.5.29 PatchMasterGroupGetComment

PatchMasterGroupGetComment returns the comment for the current group.

Syntax

```
PatchMasterGroupGetComment session, comment
```

session A Long value representing the current session.

comment A Variant variable in which the group comment is returned.

Example

```
Dim text As Variant
```

```
PatchMasterGroupGetComment session, text
```

Discussion

The current group must be set. To set the current group, use PatchMasterGroupSet. [↔ PatchMasterGroupSet, p. 223.](#)

The group comment is entered by the user, and may be empty.

7.5.30 PatchMasterGroupGetCount

PatchMasterGroupGetCount returns the number of groups in the data set.

Syntax

```
PatchMasterGroupGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of groups in the current data set.

Example

```
Dim group_count As Long
```

```
PatchMasterGroupGetCount session, count
```

Discussion

You must have a data set open.

7.5.31 PatchMasterGroupGetEperiment

PatchMasterGroupGetEperiment returns the group experiment number.

Syntax

PatchMasterGroupGetEperiment *session, experiment*

session A Long value representing the current session.

experiment A Long variable to receive the group experiment number.

Example

```
Dim group_experiment As Long

PatchMasterGroupGetEperiment session,
group_experiment
```

Discussion

The current group must be set. To set the current group, use PatchMasterGroupSet. [↗ PatchMasterGroupSet, p. 223.](#)

7.5.32 PatchMasterGroupGetLabel

PatchMasterGroupGetLabel returns the label for the current group.

Syntax

PatchMasterGroupGetLabel *session, label*

session A Long value representing the current session.

label A Variant variable to receive the group label

Example

```
Dim text As Variant

PatchMasterGroupGetLabel session, text
```

Discussion

The current group must be set. To set the current group, use PatchMasterGroupSet. [↗ PatchMasterGroupSet, p. 223.](#)

7.5.33 PatchMasterGroupGetMarked

PatchMasterGroupGetMarked returns whether the current group has been marked.

Syntax

PatchMasterGroupGetMarked *session, marked*

session A Long value representing the current session.

marked An Long variable to receive the marked state.

Example

```
Dim marked As Long
```

```
PatchMasterGroupGetMarked session, marked
```

Discussion

The current group must be set. To set the current group, use PatchMasterGroupSet. [↗ PatchMasterGroupSet, p. 223.](#)

7.5.34 PatchMasterGroupSet

PatchMasterGroupSet selects the current group.

Syntax

PatchMasterGroupSet *session, group*

session A Long value representing the current session.

group A Long value representing the group to select.

Example

The following example sets the current group to the last group in a data set.

```
Dim group_count As Long
```

```
PatchMasterGroupGetCount session, group_count
```

```
PatchMasterGroupSet session, group_count
```

Discussion

You must have a data set open.

The groups in a data set are numbered from 1 to N, where N is the value returned by PatchMasterGroupGetCount. [↗ PatchMasterGroupGetCount, p. 221.](#)

7.5.35 PatchMasterSegmentGetCount

PatchMasterSegmentGetCount returns the number of segments for the specified stimulus channel.

Syntax

```
PatchMasterSegmentGetCount session, channel, count
```

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

count A Long variable to receive the number of stimulus segments in the current series.

Example

```
Dim segment_count As Long
```

```
PatchMasterSegmentGetCount session, channel, count
```

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PatchMasterStimulusGetCount. [☞ PatchMasterStimulusGetCount, p. 233.](#)

The current series must be set. To set the current series, use PatchMasterSeriesSet. [☞ PatchMasterSeriesSet, p. 231.](#)

7.5.36 PatchMasterSegmentGetParameters

PatchMasterSegmentGetParameters returns the parameters of the specified segment for the current sweep.

Syntax

```
PatchMasterSegmentGetParameters
  session, channel, segment, type, voltage, duration
```

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

segment A Long value representing the segment to use.

type A Long variable to receive the type of the segment.

voltage A Double variable to receive the voltage of the segment.

duration A Long variable to receive the duration of the segment. The duration is measured in samples.

Example

The following example obtains the parameters of the last segment of the current series.

```
Dim segment As Long
Dim segment_type As Long
Dim voltage As Double
Dim duration As Long
```

```
PatchMasterSegmentGetCount session, channel,
  segment
```

```
If segment < 1 Then
  ' Handle the error
End If
```

```
PatchMasterSegmentGetParameters session, channel,
  segment, segment_type, voltage, duration
```

Discussion

The segment type value is interpreted according to

table 110.

Table 110 Segment types

Type	Interpretation
0	Constant
1	Ramp
2	Continuous
3	Sine wave
4	Square wave

If the segment type is constant or continuous, the voltage represents a fixed output value.

If the segment type is ramp, the voltage represents the final voltage of the segment. The initial value is the final voltage of the previous segment.

If the segment type is either sine or square wave, the output is a constant value of specified voltage plus a sine wave or square wave waveform. The parameters of the waveform may be obtained using `PatchMasterSegmentGetWaveform`.
 ☞ *PatchMasterSegmentGetWaveform*, p. 225.

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use `PatchMasterStimulusGetCount`.
 ☞ *PatchMasterStimulusGetCount*, p. 233.

You must have a sweep selected. To set the current sweep, use `PatchMasterSweepSet`.
 ☞ *PatchMasterSweepSet*, p. 246.

Segment numbers are in the range 1 to N, where N is the value returned by `PatchMasterSegmentGetCount`.
 ☞ *PatchMasterSegmentGetCount*, p. 224.

7.5.37 PatchMasterSegmentGetWaveform

`PatchMasterSegmentGetWaveform` returns the periodic waveform parameters for the specified stimulus channel.

Syntax

`PatchMasterSegmentGetWaveform session, channel, period, voltage`

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

period A Double variable to receive the waveform period, measured in seconds.

voltage A Double variable to receive the waveform voltage, measured in volts.

Example

```
Dim period As Double
Dim voltage As Double
```

```
PatchMasterSegmentGetWaveform session, channel,
period, voltage
```

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use `PatchMasterStimulusGetCount`.
 ☞ *PatchMasterStimulusGetCount*, p. 233.

The current series must be set. To set the current series, use `PatchMasterSeriesSet`.
 ☞ *PatchMasterSeriesSet*, p. 231.

The waveform parameters apply if the segment type is sine wave or square wave. You can use `PatchMasterSegmentGetParameters` to obtain the type of a segment.
 ☞ *PatchMasterSegmentGetParameters*, p. 224.

7.5.38 PatchMasterSeriesGetComment

PatchMasterSeriesGetComment returns the comment for the current series.

Syntax

PatchMasterSeriesGetComment *session, comment*

session A Long value representing the current session.

comment A Variant variable in which the series comment is returned.

Example

```
Dim text As Variant
```

```
PatchMasterSeriesGetComment session, text
```

Discussion

You must have a data set open.

The series comment is entered by the user, and may be empty.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.39 PatchMasterSeriesGetCount

PatchMasterSeriesGetCount returns the number of series in the current group.

Syntax

PatchMasterSeriesGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of series in the current group.

Example

```
Dim series_count As Long
```

```
PatchMasterSeriesGetCount session, count
```

Discussion

The current group must be set. To set the current group, use PatchMasterGroupSet. [PatchMasterGroupSet](#), p. 223.

7.5.40 PatchMasterSeriesGetLabel

PatchMasterSeriesGetLabel returns the label for the current series.

Syntax

PatchMasterSeriesGetLabel *session, label*

session A Long value representing the current session.

label A Variant variable to receive the series label

Example

Dim text As Variant

PatchMasterSeriesGetLabel session, text

Discussion

You must have a data set open.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

7.5.41 PatchMasterSeriesGetLeak

PatchMasterSeriesGetLeak returns the leak parameters for the current series.

Syntax

PatchMasterSeriesGetLeak *session, count, scaling, potential, alternate, alternate_averaging, delay*

session A Long value representing the current session.

count A Long variable to receive the number of sweeps averaged for a leak trace.

scaling A Double variable to receive size of the leak trace relative to the main trace.

potential A Double variable to receive the leak holding potential.

alternate A Long variable to receive the leak alternate protocol.

alternate_averaging A Long variable to receive the leak alternate averaging protocol.

delay A Double variable to receive the delay between the leak and the stimulus.

Example

Dim count As Long

Dim scaling As Double

Dim potential As Double

Dim alternate As Long

Dim alternate_averaging As Long

Dim delay As Double

PatchMasterSeriesGetLeak session, count, scaling,
potential, alternate, alternate_averaging, delay

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

If alternate is non-zero every second leak pulse is inverted relative to the leak holding potential.

If alternate averaging is non-zero every second leak tem-

plate is inverted relative to the leak holding potential when averaging over multiple traces.

The delay is the time between the leak pulses and the main stimulus pulse. When delay is positive, the leak pulses follow the main stimulus, otherwise the leak pulses precede the main stimulus.

7.5.42 PatchMasterSeriesGetMarked

PatchMasterSeriesGetMarked returns whether the current series has been marked.

Syntax

```
PatchMasterSeriesGetMarked session, marked
```

session A Long value representing the current session.

marked An Long variable to receive the marked state.

Example

```
Dim marked As Long
```

```
PatchMasterSeriesGetMarked session, marked
```

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

7.5.43 PatchMasterSeriesGetSampleInterval

PatchMasterSeriesGetSampleInterval returns the sampling interval in seconds for the current series.

Syntax

```
PatchMasterSeriesGetSampleInterval session, interval
```

session A Long value representing the current session.

interval A Double variable to receive the sampling interval.

Example

```
Dim interval As Double
```

```
PatchMasterSeriesGetSampleInterval session, interval
```

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

All channels are sampled at the same interval. The interval measures the time between samples on the same channel. However a given channel may be compressed. [↗ PatchMasterChannelGetSampleInterval, p. 214.](#)

7.5.44 PatchMasterSeriesGetStart

PatchMasterSeriesGetStart returns the start time in seconds for the current series.

Syntax

```
PatchMasterSeriesGetStart session, start
```

session A Long value representing the current session.

start A Double variable to receive the start time.

Example

```
Dim series_start As Double
```

```
PatchMasterSeriesGetStart session, series_start
```

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

The start time of an series is measured in seconds since the beginning of the experiment.

7.5.45 PatchMasterSeriesGetTime

PatchMasterSeriesGetTime returns the time of the start of the current series.

Syntax

```
PatchMasterSeriesGetTime session, date_time
```

session A Long value representing the current session.

date_time A Date variable to receive the series start time.

Example

```
Dim date_time As Date
```

```
PatchMasterSeriesGetTime session, date_time
```

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

7.5.46 PatchMasterSeriesGetUserName

PatchMasterSeriesGetUserName returns the user name supplied for the current series.

Syntax

```
PatchMasterSeriesGetUserName session, name
```

session A Long value representing the current session.

name A Variant variable to receive the user name

Example

```
Dim name As Variant
```

```
PatchMasterSeriesGetUserName session, name
```

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

7.5.47 PatchMasterSeriesGetUserParameter

PatchMasterSeriesGetUserParameter returns the first user parameter for the current series.

Syntax

PatchMasterSeriesGetUserParameter *session, index, name., value, units*

session A Long value representing the current session.

index A Long value specifying which parameter to return.

name A Variant variable to receive the **parameter name**.

value A Double variable to receive the parameter value.

units A Variant variable to receive the parameter units.

Example

```
Dim name As Variant
Dim parameter As Double
Dim units As Variant
```

```
PatchMasterSeriesGetUserParameter session, 0, name,
parameter, units
```

Discussion

The index can range between 0 and 3.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.48 PatchMasterSeriesSet

PatchMasterSeriesSet selects the current series.

Syntax

PatchMasterSeriesSet *session, series*

session A Long value representing the current session.

series A Long value representing the number of the series to use within the current group.

Example

The following example sets the current series to the last series in the current group.

```
Dim series As Long
```

```
PatchMasterSeriesGetCount session, series
```

```
If series < 1 Then
    ' Handle the error
End If
```

```
PatchMasterSeriesSet session, series
```

Discussion

Series are numbered 1 to N, where N is the number of series in the current group. The number of series in the current group is returned by PatchMasterSeriesGetCount. [PatchMasterSeriesGetCount](#), p. 226.

7.5.49 PatchMasterSessionClose

PatchMasterSessionClose ends the session. It deallocates any resources allocated DataAccess.

Syntax

```
PatchMasterSessionClose session
```

session A Long value representing the current session.

Example

```
PatchMasterFileClose session
```

```
PatchMasterSessionClose session
```

Discussion

PatchMasterSessionClose does not return a value, so it should not be called as a function.

You should close a session to deallocate any resources allocated to the session.

7.5.50 PatchMasterSessionOpen

PatchMasterSessionOpen begins a DataAccess session.

Syntax

```
PatchMasterSessionOpen session
```

session A Long variable to contain the created session.

Example

```
Dim session As Long
Dim result As Long
Dim file_name As Variant
file_name = "Sample.dat"
```

```
PatchMasterSessionOpen session
```

```
result = PatchMasterFileOpen(session, file_name)
```

Discussion

You must have a session open to use any other DataAccess calls. The handle returned by PatchMasterSessionOpen is a parameter to all other calls.

You can have as many sessions open simultaneously as you would like. For example, if you want to have two data sets open simultaneously, call PatchMasterSessionOpen twice. Use one handle when accessing one data set and the other handle when accessing the other data set.

After you are done with the session, you should call PatchMasterSessionClose. [☞ PatchMasterSessionClose, p. 232.](#)

7.5.51 PatchMasterStimulusGetCount

PatchMasterStimulusGetCount returns the number of stimulus channels used for the current series.

Syntax

```
PatchMasterStimulusGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of stimulus channels used in the current series.

Example

```
Dim count As Long
```

```
PatchMasterStimulusGetCount session, count
```

Discussion

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

7.5.52 PatchMasterStimulusGetDAC

PatchMasterStimulusGetDAC obtains the physical D/A channel.

Syntax

```
PatchMasterStimulusGetDAC session, channel, dac
```

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

dac An Long variable to receive the D/A channel number.

Example

```
Dim dac As Long
```

```
PatchMasterStimulusGetDAC session, dac
```

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PatchMasterStimulusGetCount. [↗ PatchMasterStimulusGetCount, p. 233.](#)

The current series must be set. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

The physical channel number is the physical channel label on the data acquisition device.

7.5.53 PatchMasterStimulusGetRange

PatchMasterStimulusGetRange returns the range of the stimulus.

Syntax

```
PatchMasterStimulusGetRange session, minimum,  
maximum
```

session A Long value representing the current session.

minimum A Double variable to receive the lower limit.

maximum A Double variable to receive the upper limit.

Example

```
Dim minimum As Double  
Dim maximum As Double
```

```
PatchMasterStimulusGetRange session, minimum,  
maximum
```

Discussion

You must have a data set open.

PatchMasterStimulusGetRange returns the minimum and maximum possible values for the stimulus.

7.5.54 PatchMasterStimulusGetRelevantSegments

PatchMasterStimulusGetRelevantSegments returns the relevant x and y segments for the specified stimulus channel.

Syntax

```
PatchMasterStimulusGetRelevantSegments session,  
channel, x_segment, y_segment
```

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

x_segment A Long variable to receive the relevant x segment.

y_segment A Long variable to receive the relevant y segment.

Example

```
Dim relevant_x As Long  
Dim relevant_y As Long
```

```
PatchMasterStimulusGetRelevantSegments  
session, channel, relevant_x, relevant_y
```

Discussion

The relevant x segment is the stimulus segment containing the relevant independent variable. The relevant y segment is the acquired data segment containing the “interesting” data.

The returned segment numbers will be in the range 1 to N, where N is the value returned by PatchMasterSegmentGetCount. [☞ PatchMasterSegmentGetCount, p. 224.](#)

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PatchMasterStimulusGetCount. [☞ PatchMasterStimulusGetCount, p. 233.](#)

The current series must be set. To set the current series, use PatchMasterSeriesSet. [☞ PatchMasterSeriesSet, p. 231.](#)

7.5.55 PatchMasterStimulusGetSource

PatchMasterStimulusGetSource returns the stimulus source for the specified stimulus channel.

Syntax

PatchMasterStimulusGetSource *channel, source*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

source A Long variable to receive the stimulus source.

Example

Dim source As Long

PatchMasterStimulusGetSource session, 0, source

Discussion

The stimulus source is one of the values shown in table 111.

Table 111 Stimulus source

Source	Description
0	Template
1	File

If the source is 0, the stimulus was generated according to the stimulus protocol. If the source is 1, the stimulus came from a separate binary file containing an arbitrary waveform.

The current series must be set. To set the current series, use PMSeriesSet. [PatchMasterSeriesSet, p. 231.](#)

7.5.56 PatchMasterStimulusGetStartSegment

PatchMasterStimulusGetStartSegment returns the starting segment and sample for the input channel tied to the specified stimulus channel.

Syntax

PatchMasterStimulusGetStartSegment *session, channel, segment, sample*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

segment A Long variable to receive the starting segment.

sample A Long variable to receive the start sample.

Example

Dim segment As Long

Dim sample As Long

PatchMasterStimulusGetStartSegment
session, channel, segment, sample

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PatchMasterStimulusGetCount. [PatchMasterStimulusGetCount, p. 233.](#)

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet, p. 231.](#)

7.5.57 PatchMasterStimulusGetTriggerType

PatchMasterStimulusGetTriggerType returns the trigger type for the current series.

Syntax

PatchMasterStimulusGetTriggerType *session, type*

session A Long value representing the current session.

type A Long variable to receive the trigger type.

Example

Dim type As Long

PatchMasterStimulusGetTriggerType session, type

Discussion

The trigger type is one of the values shown in table 112.

Table 112 Trigger types

Channel	Output channel
0	None
1	Series
2	Sweep
3	Sweep no leak

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.58 PatchMasterStimulusGetUnits

PatchMasterStimulusGetUnits returns the units for the specified stimulus channel.

Syntax

PatchMasterStimulusGetUnits *session, channel, units*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

units A Variant variable to receive the units.

Example

Dim stimulus_units As Variant

PatchMasterStimulusGetUnits session, channel, stimulus_units

Discussion

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PatchMasterStimulusGetCount. [PatchMasterStimulusGetCount](#), p. 233.

The current series must be set. To set the current series, use PatchMasterSeriesSet. [PatchMasterSeriesSet](#), p. 231.

7.5.59 PatchMasterSweepGetCount

PatchMasterSweepGetCount returns the number of sweeps in the current series.

Syntax

PatchMasterSweepGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of sweeps recorded in the current series.

Example

```
Dim sweeps As Long
```

```
PatchMasterSweepGetCount session, sweeps
```

Discussion

You must have a series selected. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in the series.

This count may be less than the number of sweeps originally recorded if the data set has been edited. [↗ PatchMasterSweepGetRecordedCount, p. 239.](#)

7.5.60 PatchMasterSweepGetDigitalIn

PatchMasterSweepGetDigitalIn the digital input value for the current sweep.

Syntax

PatchMasterSweepGetDigitalIn *session, value*

session A Long value representing the current session.

value A Long value to receive the digital value.

Example

```
Dim value As Long
```

```
PatchMasterSweepGetDigitalIn session, value
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

7.5.61 PatchMasterSweepGetHoldingPotential

PatchMasterSweepGetHoldingPotential returns the holding potential for the current sweep for the specified stimulus channel.

Syntax

PatchMasterSweepGetHoldingPotential *session, channel, potential*

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

potential A Double variable to the sweep holding potential.

Example

Dim potential As Double

PatchMasterSweepGetHoldingPotential session,
channel, potential

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PatchMasterStimulusGetCount. [↗ PatchMasterStimulusGetCount, p. 233.](#)

7.5.62 PatchMasterSweepGetLabel

PatchMasterSweepGetLabel returns the label for the current sweep.

Syntax

PatchMasterSweepGetLabel *session, label*

session A Long value representing the current session.

label A Variant variable to receive the sweep label

Example

Dim text As Variant

PatchMasterSweepGetLabel session, text

Discussion

You must have a data set open.

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

7.5.63 PatchMasterSweepGetMarked

PatchMasterSweepGetMarked returns whether the current sweep has been marked.

Syntax

PatchMasterSweepGetMarked *session, marked*

session A Long value representing the current session.

marked An Long variable to receive the marked state.

Example

Dim marked As Long

PatchMasterSweepGetMarked session, marked

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

7.5.64 PatchMasterSweepGetRecordedCount

PatchMasterSweepGetRecordedCount returns the number of sweeps originally recorded for the current series.

Syntax

PatchMasterSweepGetRecordedCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of sweeps recorded in the current series.

Example

Dim sweeps As Long

PatchMasterSweepGetRecordedCount session, sweeps

Discussion

You must have a series selected. To set the current series, use PatchMasterSeriesSet. [↗ PatchMasterSeriesSet, p. 231.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in the series.

This count may be greater than the number of sweeps currently available if the data set has been edited. [↗ PatchMasterSweepGetCount, p. 237.](#)

7.5.65 PatchMasterSweepGetSampleCount

PatchMasterSweepGetSampleCount returns the number of samples in the current sweep for the specified channel.

Syntax

```
PatchMasterSweepGetSampleCount session, channel,
    samples
```

session A Long value representing the current session.

channel A Long value representing the channel.

samples A Long variable to receive the number of samples recorded in the current sweep.

Example

```
Dim samples As Long
```

```
PatchMasterSweepGetSampleCount session, channel,
    samples
```

Discussion

The number of samples returned by PatchMasterSweepGetSampleCount is the number of samples recorded for the specified channel.

The sample count for different channels may differ if they are tied to different stimulus channels.

The number of samples recorded may be less than the total duration of all stimulus segments if recording does not begin at the beginning of the first stimulus segment. [☞ PatchMasterStimulusGetStartSegment, p. 235.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. To determine the number of channels, use PatchMasterChannelGetCount. [☞ PatchMasterChannelGetCount, p. 210.](#)

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [☞ PatchMasterSweepSet, p. 246.](#)

7.5.66 PatchMasterSweepGetSolutions

PatchMasterSweepGetSolutions returns the solution numbers for the current sweep.

Syntax

```
PatchMasterSweepGetSolutions session, internal, external
```

session A Long value representing the current session.

internal A Long variable to receive the internal solution number.

external A Long variable to receive the external solution number.

Example

```
Dim internal As Long
Dim external As Long
```

```
PatchMasterSweepGetSolutions session, internal,
    external
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [☞ PatchMasterSweepSet, p. 246.](#)

7.5.67 PatchMasterSweepGetStart

PatchMasterSweepGetStart returns the sweep start time.

Syntax

```
PatchMasterSweepGetStart session, sweep_time, data_time
```

session A Long value representing the current session.

sweep_time A Double variable to receive the starting time of the sweep.

data_time A Double variable to receive the starting time of the sweep data.

Example

```
Dim sweep_time As Double
Dim data_time As Double
```

```
PatchMasterSweepGetStart session, sweep_time,
data_time
```

Discussion

The sweep time is measured in seconds since the beginning of the series. This is the time the stimulus starts.

The data time is measured in seconds since the beginning of the series. This is the starting time of recorded data. [↗ PatchMasterStimulusGetStartSegment, p. 235.](#)

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

7.5.68 PatchMasterSweepGetTemperature

PatchMasterSweepGetTemperature returns the temperature for the current series.

Syntax

```
PatchMasterSweepGetTemperature session, temperature
```

session A Long value representing the current session.

temperature A Double variable to receive the **temperature**.

Example

```
Dim temperature As Double
```

```
PatchMasterSweepGetTemperature session, temperature
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

7.5.69 PatchMasterSweepGetTimer

PatchMasterSweepGetTimer returns the timer value at the start of the sweep.

Syntax

```
PatchMasterSweepGetTimer session, timer
```

session A Long value representing the current session.

timer A Double variable to receive the timer value at the start of the sweep.

Example

```
Dim timer As Double
```

```
PatchMasterSweepGetTimer session, timer
```

Discussion

The timer value is measured in seconds since the most recent timer reset.

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

7.5.70 PatchMasterSweepGetUserParameter

PatchMasterSweepGetUserParameter returns the first user parameter for the current series.

Syntax

```
PatchMasterSweepGetUserParameter session, index, name, value, units
```

session A Long value representing the current session.

index A Long value specifying which parameter to return.

name A Variant variable to receive the **parameter name**.

value A Double variable to receive the parameter value.

units A Variant variable to receive the parameter units.

Example

```
Dim name As Variant
Dim parameter As Double
Dim units As Variant
```

```
PatchMasterSweepGetUserParameter session, 0, name, parameter, units
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [↗ PatchMasterSweepSet, p. 246.](#)

The index can range between 0 and 3.

7.5.71 PatchMasterSweepRead

PatchMasterSweepRead reads the data for a section of one channel of the current sweep into an array.

Syntax

```
PatchMasterSweepRead session, channel, start, length,
    data
```

session A Long value representing the current session.

channel A Long value representing the channel containing the data to read.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

data An array of "Double" to receive the data. The array must be long enough to contain all the data in the sweep.

Example

```
Dim samples As Long
Dim buffer() As Double
PatchMasterSweepGetSampleCount session, channel,
    samples

If samples > limit Then
    samples = limit
End If

ReDim buffer(samples)

PatchMasterSweepRead session, channel, 0, samples,
    buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet](#), p. 246.

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [PatchMasterChannelGetCount](#), p. 210.

You can use PatchMasterChannelGetAvailable to determine if leak data is available for the current sweep.

[PatchMasterChannelGetAvailable](#), p. 207.

The number of samples in the current sweep is provided by PatchMasterSweepGetSampleCount.

[PatchMasterSweepGetSampleCount](#), p. 240.

7.5.72 PatchMasterSweepReadLeak

PatchMasterSweepReadLeak reads the leak template for the specified channel of the current sweep into an array.

Syntax

```
PatchMasterSweepReadLeak session, channel, start, length,
    data
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

data An array of “Double” to receive the data. The array must be long enough to contain all the data in the leak template.

Example

```
Dim samples As Long
Dim buffer() As Double
```

```
PatchMasterSweepGetSampleCount session, channel,
    samples
```

```
ReDim buffer(samples)
```

```
PatchMasterSweepReadLeak session, channel, 0,
    samples, buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [PatchMasterSweepSet, p. 246.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [PatchMasterChannelGetCount, p. 210.](#)

You can use PatchMasterChannelGetLeakAvailable to determine if leak data is available for the current sweep. [PatchMasterChannelGetLeakAvailable, p. 211.](#)

The number of samples in the current sweep is provided by

PatchMasterSweepGetSampleCount.

[PatchMasterSweepGetSampleCount, p. 240.](#)

7.5.73 PatchMasterSweepReadRaw

PatchMasterSweepRead reads the data without leak correction for the specified channel of the current sweep into an array.

Syntax

```
PatchMasterSweepReadRaw session, channel, start, length,
    data
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

data An array of “Double” to receive the data. The array must be long enough to contain all the data in the sweep.

Example

```
Dim samples As Long
Dim buffer() As Double
PatchMasterSweepGetSampleCount session, samples
```

```
If samples > limit Then
    samples = limit
End If
```

```
ReDim buffer(samples)
```

```
PatchMasterSweepReadRaw session, 0, samples,
buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [☞ PatchMasterSweepSet, p. 246.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [☞ PatchMasterChannelGetCount, p. 210.](#)

You can use PatchMasterChannelGetAvailable to determine if leak data is available for the current sweep. [☞ PatchMasterChannelGetAvailable, p. 207.](#)

7.5.74 PatchMasterSweepReadStimulus

PatchMasterSweepReadStimulus reads the specified stimulus channel of the current sweep into an array.

Syntax

```
PatchMasterSweepReadStimulus session, channel, start,
    length, stimulus
```

session A Long value representing the current session.

channel A Long value representing the stimulus channel to use.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

stimulus An array of “Double” to receive the stimulus. The array must be long enough to contain all the data in the sweep stimulus.

Example

```
Dim samples As Long
Dim buffer() As Double
```

```
PatchMasterSweepGetSampleCount session, 0, samples
```

```
ReDim buffer(samples)
```

```
PatchMasterSweepReadStimulus session, channel, 0,
samples, buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PatchMasterSweepSet. [☞ PatchMasterSweepSet, p. 246.](#)

Stimulus channels are numbered 0 to N-1, where N is the number of stimulus channels in the series. To determine the number of stimulus channels, use PatchMasterStimulusGetCount. [☞ PatchMasterStimulusGetCount, p. 233.](#)

The number of samples in the current sweep is provided by PatchMasterSweepGetSampleCount. [☞ PatchMasterSweepGetSampleCount, p. 240.](#)

7.5.75 PatchMasterSweepSet

PatchMasterSweepSet selects the current sweep.

Syntax

```
PatchMasterSweepSet session, sweep
```

session A Long value representing the current session.

sweep A Long value representing the sweep to set.

Example

This example sets the current sweep to the last sweep in the current series.

```
Dim sweeps As Long
```

```
PatchMasterSweepGetCount session, sweeps
```

```
If sweeps < 1 Then
```

```
    ' Handle the error
```

```
End If
```

```
PatchMasterSweepSet session, sweeps
```

Discussion

Sweeps are numbered 1 to N, where N is the number of sweeps in the current series. The number of sweeps in the current series is provided by PatchMasterSweepGetCount. [☞ PatchMasterSweepGetCount, p. 237.](#)

You must have a series selected. To set the current series, use PatchMasterSeriesSet. [☞ PatchMasterSeriesSet, p. 231.](#)

8 Pulse: IGOR Pro

Section	
8.1 Script Interface	P. 247
8.2 Robust Processing	P. 250
8.3 Operations	P. 251
8.4 Reference	P. 252

8.1 Script Interface

This section explains how to access a Pulse data file. It contains a step by step description of the operations to perform to open a file and read the data in it.

8.1.1 Opening a Data Set

Open an Pulse data set using the PulseFileOpen operation.

☞ *PulseFileOpen, p. 257.*

In order to open a data set, you must supply the full path to the DAT file. The IGOR PRO “Open” command can be used to bring up a file selection dialog and supply the resulting file path.

The following example brings up a file selection dialog to obtain a file specification. It then uses PulseFileOpen to open the data set for processing:

```
Variable fileReferenceNumber
Variable status

Open /D/R/T="BINA" fileReferenceNumber

if (strlen(S_fileName) > 0)
    PulseFileOpen S_fileName, status
endif
```

8.1.2 File Parameters

Once you have opened a data set, you can determine the parameters of the data set. The parameters include the creation date and time and the data set comment. ☞ *File Parameter Operations, p. 251.*

The following example obtains data set parameters:

```
Variable status
String fileDateTime
PulseFileGetTime fileDateTime, status
Print fileDateTime
```

```
String fileComment
PulseFileGetComment fileComment, status
Print fileComment
```

8.1.3 Groups

A data set is a sequence of groups. You use the group operations to select a group and determine its parameters. ☞ *Group Operations, p. 251.*

You use PulseGroupGetCount to determine the number of groups in a data set. ☞ *PulseGroupGetCount, p. 259.*

Groups are numbered 1 to N, where N is the number of groups in a data set. You use PulseGroupSet to set the group to use. ☞ *PulseGroupSet, p. 261.*

The following example prints the label of each group in a data set. The label is a text string.

```
Variable group_count
Variable current_group
String group_label
Variable status
```

```
PulseGroupGetCount group_count, status
```

```
if (group_count > 0)
    current_group = 1
```

```

do
  PulseGroupSet current_group, status
  PulseGroupGetLabel group_label, status
  Print group_label
  current_group += 1
  while (current_group <= group_count)
else
  | The data set contains no groups
endif

```

A file may have no groups if no data was acquired.

8.1.4 Series

A group is a sequence of series. You use the series operations to select a series and determine its parameters. [☞ Series Operations, p. 251.](#)

You use `PulseSeriesGetCount` to determine the number of series in a group. [☞ PulseSeriesGetCount, p. 266.](#)

Series are numbered 1 to N, where N is the number of series in a group. You use `PulseSeriesSet` to set the series to use. [☞ PulseSeriesSet, p. 274.](#)

The following example prints the label of each series in the current group. The label is a text string.

```

Variable series_count
Variable current_series
String series_label
Variable status

PulseSeriesGetCount series_count, status

if (series_count > 0)
  current_series = 1
  do
    PulseSeriesSet current_series, status
    PulseSeriesGetLabel series_label, status
    Print series_label
    current_series += 1
  while (current_series <= series_count)
else
  | The group contains no series
endif

```

A group may have no series if no data was acquired.

8.1.5 Sweeps

A series is a sequence of sweeps. You use the sweep operations to select a sweep and determine its parameters. [☞ Sweep Operations, p. 252.](#)

You use `PulseSweepGetCount` to determine the number of sweeps in a sweep. [☞ PulseSweepGetCount, p. 278.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in a series. You use `PulseSweepSet` to set the sweep to use. [☞ PulseSweepSet, p. 285.](#)

The following example prints the number of samples in each sweep in the current series.

```

Variable sweep_count
Variable current_sweep
Variable sweep_samples
Variable status

PulseSweepGetCount sweep_count, status

if (sweep_count > 0)
  current_sweep = 1
  do
    PulseSweepSet current_sweep, status
    PulseSweepGetSampleCount sweep_samples,
      status
    Print sweep_samples
    current_sweep += 1
  while (current_sweep <= sweep_count)
else
  | The series contains no sweep
endif

```

A series may have no sweep if no data was acquired.

8.1.6 Segments

A stimulus template is described by a sequence of segments. Although all sweeps in a series use the same stimulus template, the duration and voltage of each segment can vary with each sweep.

Segment numbers range from 1 to N, where N is the number of segments in the series. You use `PulseSegmentGetCount` to determine the number of segments in a series. [☞ PulseSegmentGetCount, p. 261.](#)

8.1.7 Channels

PulseChannelGetCount returns the number of channels recorded in a sweep. Normally the number of channels recorded in each sweep in a series will be the same when a data set is acquired, but channels can be removed from individual sweeps during data set editing, leaving a series with varying numbers of channels recorded in each sweep. [☞ PulseChannelGetCount, p. 253.](#)

The following example obtains the number of channels recorded in the current sweep:

```
Variable channel_count
Variable status

PulseChannelGetCount channel_count, status
```

Channel numbers range from 0 to N-1, where N is the number of channels in the sweep. Existing versions of Pulse never record more than two channels in a sweep.

8.1.8 Reading Data

Normally one would read whole sweeps but it may be necessary to read sections of a sweep if you have insufficient memory available to read whole sweeps.

The wave into which you read data must contain double-precision values.

Reading Sweeps

To read an entire sweep, use PulseSweepRead. [☞ PulseSweepRead, p. 283.](#)

The following example reads each sweep of a series into a wave.

```
Variable sweep_count
Variable current_sweep
Variable samples
Variable status

PulseSweepGetCount sweep_count, status

if (sweep_count > 0)
  current_sweep = 1
  do
    PulseSweepSet current_sweep, status
    PulseSweepGetSampleCount samples, status
```

```
Make/D/N=(samples) data
```

```
PulseSweepRead channel, 0, samples, data, status
| Data processing goes here
current_sweep += 1
while (current_sweep <= sweep_count)
else
| The series has no sweeps
endif
```

Sweeps and Channels

Each channel in an sweep has the same number of data samples.

The following example reads data from each channel of the current sweep into a wave.

```
Variable channel_count
Variable channel
Variable samples
Variable status

PulseSweepGetSampleCount samples, status

Make/D/N=(samples) data

PulseChannelGetCount channel_count, status

channel = 0
do
  PulseSweepRead channel, 0, samples, data, status
  | Data processing goes here
  channel += 1
while (channel < channel_count)
```

Reading Sections of a Sweep

To read sections of a sweep, use PulseSweepRead. [☞ PulseSweepRead, p. 283.](#)

The following example reads the first “length” values of the first sweep into an array.

```
Variable samples
Variable status

PulseSweepGetSampleCount samples, status

if (samples > length)
{
```

```

    samples = length;
};

Make/D/N=(samples) data

PulseSweepRead channel, 0, samples, data, status

```

8.1.9 Closing a Data Set

To close a data set, use `PulseFileClose`. [☞ *PulseFileClose*, p. 255.](#)

The following example closes a Pulse data set.

```
PulseFileClose
```

8.2 Robust Processing

If you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

This section explains how to use the information provided by Data Access Pro to recover from error conditions in your IGOR Pro procedures.

8.2.1 Errors

The status information provided by Data Access Pro allows you to recover from error conditions caused by the program, user or the system. It does not allow you to recover from syntax errors in your program.

Program Errors

Errors in your program include such mistakes as using `PulseSeriesSet` to set an invalid series number. You can determine the range of valid series numbers using `PulseSeriesGetCount`, so you can avoid setting an invalid segment number. Catching and reporting such errors is helpful in debugging a procedure.

System Errors

System errors include such problems as an error reading a Pulse data file. Such errors are usually the result of prob-

lems such as a corrupted disk, a missing floppy or CD-ROM, or defective hardware such as a disk drive. Catching such errors prevents further problems and may allow the procedure to recover, for example by allowing the user to insert a missing file medium.

User Errors

User errors include mistakes such as when the user selects a file that is not a Pulse data file. The recovery allows your procedures to provide an error indication to the user and allow the user to select a different file.

Syntax Errors

Syntax errors include mistakes such as passing the wrong number or types of parameters to an operation. An improperly dimensioned wave is also treated as a syntax error. Such errors are returned directly to Igor Pro and cause a procedure to terminate.

8.2.2 Status

Nearly all operations return a status value through a status parameter. The status value is non-zero if an error occurred. All Data Access and system errors are returned in this manner. Such errors will not terminate a procedure. The procedure should be designed to handle such conditions. Igor Pro errors, such as a missing parameter, are returned directly to Igor and will terminate a procedure.

Status values for specific errors are not defined, and may change from version to version of Data Access Pro. The `PulseGetStatusText` operation translates a status value to text. [☞ *PulseGetStatusText*, p. 258.](#)

The following example allows the user to select a file, and does not terminate until the user selects a valid Pulse file.

```

Variable fileNumber
Variable status
String message

do
    Open /D/R/T="BINA" fileNumber

    if (strlen(S_fileName) > 0)
        PulseFileOpen S_fileName, status

        if (status != 0)

```

```

PulseGetStatusText status, message
Print "PulseFileOpen error: ", message
endif
endif
while (status != 0)

```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

8.3 Operations

This chapter describes each of the operations provided by DataAccess Pro. The operations are grouped by category.

8.3.1 Common Operations

The common operations are valid regardless of whether a data set is open or not. The common operations are shown in table 113.

Table 113 Common operations

Operation	
PulseGetVersion	p. 258
PulseGetStatusText	p. 258

8.3.2 File Operations

The file operations allow you to access a Pulse data set. The file operations are shown in table 114.

Table 114 File operations

Operation	
PulseFileOpen	p. 257
PulseFileClose	p. 255

8.3.3 File Parameter Operations

The file parameter operations allow you to obtain data set parameters. The file parameter operations are shown in

table 115.

Table 115 File parameter operations

Operation	
PulseFileGetComment	p. 255
PulseFileGetPulseVersion	p. 256
PulseFileGetTime	p. 256
PulseFileGetVersion	p. 257

The file parameter operations are valid when a data set is open.

8.3.4 Group Operations

The group operations allow you to access each group of a data set. The group operations are shown in table 116.

Table 116 Group operations

Operation	
PulseGroupGetComment	p. 259
PulseGroupGetCount	p. 259
PulseGroupGetExperiment	p. 260
PulseGroupGetLabel	p. 260
PulseGroupSet	p. 261

The group operations are valid when a data set is open. Most require that a current group be set.

8.3.5 Series Operations

The series operations allow you to access each series of a group. The series operations are shown in table 117.

Table 117 Series operations

Operation	
PulseSeriesGetBackgroundNoise	p. 264
PulseSeriesGetBandwidth	p. 265
PulseSeriesGetCellPotential	p. 265
PulseSeriesGetComment	p. 266
PulseSeriesGetCount	p. 266
PulseSeriesGetHoldingPotential	p. 267
PulseSeriesGetLabel	p. 267
PulseSeriesGetLeak	p. 268
PulseSeriesGetPipette	p. 269
PulseSeriesGetPotentialType	p. 269
PulseSeriesGetRecordingMode	p. 270

Table 117 Series operations

Operation	
PulseSeriesGetSampleInterval	p. 270
PulseSeriesGetSealResistance	p. 271
PulseSeriesGetSolutions	p. 271
PulseSeriesGetStart	p. 272
PulseSeriesGetTemperature	p. 272
PulseSeriesGetTime	p. 273
PulseSeriesGetUserParameter1	p. 273
PulseSeriesGetUserParameter2	p. 274
PulseSeriesSet	p. 274

The series operations are valid when a data set is open. Most require that a current series be set.

8.3.6 Sweep Operations

The sweep operations allow you to access each sweep of a series. The sweep operations are shown in table 118.

Table 118 Sweep operations

Operation	
PulseSweepGetAmplifier	p. 278
PulseSweepGetCount	p. 278
PulseSweepGetHoldingPotential	p. 279
PulseSweepGetLabel	p. 279
PulseSweepGetLeakAvailable	p. 280
PulseSweepGetSampleCount	p. 281
PulseSweepGetStart	p. 281
PulseSweepGetTimer	p. 282
PulseSweepGetUserParameter1	p. 282
PulseSweepGetUserParameter2	p. 283
PulseSweepRead	p. 283
PulseSweepReadLeak	p. 284
PulseSweepReadRaw	p. 284
PulseSweepReadStimulus	p. 285
PulseSweepSet	p. 285

The sweep operations are valid when a data set is open. Most require that a current sweep be set.

8.3.7 Channel Operations

The channel operations allow you to obtain information about the acquired data channels. The channel operations

are shown in table 119.

Table 119 Channel operations

Operation	
PulseChannelGetCount	p. 253
PulseChannelGetADC	p. 253
PulseChannelGetRange	p. 254
PulseChannelGetUnits	p. 254

8.3.8 Stimulus Operations

The stimulus operations allow you to obtain information about the stimulus protocol. The stimulus operations are shown in table 120.

Table 120 Stimulus operations

Operation	
PulseStimulusGetDAC	p. 275
PulseStimulusGetRange	p. 275
PulseStimulusGetTrigger	p. 276
PulseStimulusGetTriggerCount	p. 277
PulseStimulusGetUnits	p. 277

8.3.9 Segment Operations

The segment operations allow you to access information about the stimulus segments. The segment operations are shown in table 121.

Table 121 Segment operations

Operation	
PulseSegmentGetCount	p. 261
PulseSegmentGetParameters	p. 262
PulseSegmentGetRelevant	p. 263
PulseSegmentGetWaveform	p. 264
PulseSegmentGetTrigger	p. 263

8.4 Reference

This section lists each DataAccess Pro operation in alphabetical order.

8.4.1 PulseChannelGetADC

PulseChannelGetADC obtains the physical A/D channel.

Syntax

PulseChannelGetADC *channel, adc, status*

channel A value representing the channel.

adc A variable to receive the A/D channel number.

status A variable to receive the status of the operation.

Example

Variable adc
Variable status

PulseChannelGetADC 0, adc, status

Discussion

The logical channel number may differ from the physical A/D channel used. The physical channel number is the one labeled on the data acquisition device.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. To determine the number of channels, use PulseChannelGetCount.
☞ *PulseChannelGetCount*, p. 253.

8.4.2 PulseChannelGetCount

PulseChannelGetCount returns the number of channels recorded in the current sweep.

Syntax

PulseChannelGetCount *count, status*

count A variable to receive the number of channels recorded in the current sweep.

status A variable to receive the status of the operation.

Example

Variable status
Variable channels

PulseChannelGetCount channels, status

Discussion

The number of channels recorded may vary from sweep to sweep within a series.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep.

To set the current sweep, use PulseSweepSet.
☞ *PulseSweepSet*, p. 285.

8.4.3 PulseChannelGetRange

PulseChannelGetRange returns the range of the specified channel.

Syntax

```
PulseChannelGetRange channel, minimum, maximum,
                    status
```

channel A value representing the channel to use.

minimum A variable to receive the lower limit.

maximum A variable to receive the upper limit.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable minimum
Variable maximum
```

```
PulseChannelGetRange 0, minimum, maximum, status
```

Discussion

PulseChannelGetRange returns the minimum and maximum possible values for the specified channel.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. To determine the number of channels, use `PulseChannelGetCount`. [☞ PulseChannelGetCount, p. 253.](#)

The current sweep must be set. To set the current sweep, use `PulseSweepSet`. [☞ PulseSweepSet, p. 285.](#)

8.4.4 PulseChannelGetUnits

PulseChannelGetUnits returns the units associated with the specified channel.

Syntax

```
PulseChannelGetUnits channel, units, status
```

channel A value specifying the channel for which to return the units.

units A String variable to contain the units.

status A variable to receive the status of the operation.

Example

```
Variable status
String units

PulseChannelGetUnits channel, units, status
```

Discussion

The units for a given channel channels recorded may vary from series to series within a group.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. [☞ PulseChannelGetCount, p. 253.](#)

The current series must be set. To set the current series, use `PulseSeriesSet`. [☞ PulseSeriesSet, p. 274.](#)

8.4.5 PulseFileClose

PulseFileClose terminates processing of the Pulse data set.

Syntax

PulseFileClose

status A variable to receive the status of the operation.

Example

PulseFileClose

Discussion

You must have a data set open.

You should perform PulseFileClose as soon as you complete processing of a data set.

8.4.6 PulseFileGetComment

PulseFileGetComment returns the comment for the data set.

Syntax

PulseFileGetComment *comment, status*

comment A String variable in which the data set comment is returned.

status A variable to receive the status of the operation.

Example

Variable status

String text

PulseGetFileCommenttext, status

Discussion

You must have a data set open.

8.4.7 PulseFileGetPulseVersion

PulseFileGetPulseVersion returns the version of Pulse which created the data set.

Syntax

PulseFileGetPulseVersion *version, status*

version A String variable in which the Pulse version is returned.

status A variable to receive the status value.

Example

Variable status
String version

PulseFileGetVersion version, status

Discussion

You must have a file open.

8.4.8 PulseFileGetTime

PulseFileGetTime returns the creation date and time of the data set.

Syntax

PulseFileGetTime *date_time, status*

date_time A String variable to receive the data set creation date and time.

status A variable to receive the status of the operation.

Example

Variable status
String date_time

PulseFileGetTime date_time, status

Discussion

You must have a data set open.

8.4.9 PulseFileGetVersion

PulseFileGetVersion returns the Pulse format version of the file.

Syntax

PulseFileGetVersion *version, status*

version A String variable in which the file version is returned.

status A variable to receive the status value.

Example

Variable status

String version

PulseFileGetVersion version, status

Discussion

You must have a file open.

8.4.10 PulseFileOpen

PulseFileOpen prepares a Pulse data set for processing.

Syntax

PulseFileOpen *path, status*

path A String value representing the data set path.

status A variable to receive the status of the operation.

status A variable to receive the status of the operation.

Example

Variable status

PulseFileOpen "Data.dat", status

Discussion

You must not have a data set open.

A Pulse data set consists of three files, with the extensions PUL, PGF, and DAT. All three files must be present in the same directory for PulseFileOpen to succeed.

Once a data set is open, you can access data set parameters. [☞ File Parameter Operations, p. 251](#). You can process the groups in the data set. [☞ Group Operations, p. 251](#).

When you are done with a data set, you should close it. [☞ PulseFileClose, p. 255](#).

PulseFileOpen invalidates the current group. You must set the current group using PulseGroupSet. [☞ PulseGroupSet, p. 261](#).

8.4.11 PulseGetStatusText

PulseGetStatusText translates a status value returned by a DataAccess Pro operation to a text string.

Syntax

PulseGetStatusText *status, message*

status A value representing the status value to translate.

message A String variable in which the translated text is returned.

status A variable to receive the status of the operation.

Example

Variable status
String message

PulseFileOpen "Sample.dat", status

```
if (status <> 0)
  PulseGetStatusText status, message
  Print "PulseFileOpen error: ", message
endif
```

Discussion

You need not have a data set open.

PulseGetStatusText is the only means provided to interpret status values.

8.4.12 PulseGetVersion

PulseGetVersion returns the version number of the DataAccess Pro package.

Syntax

PulseGetVersion *version*

version A variable to receive the DataAccess Pro version number.

status A variable to receive the status of the operation.

Example

Variable version

PulseGetVersion version

Discussion

You need not have a data set open.

If you are writing a set of procedures using DataAccess Pro, you can use PulseGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess Pro. You can use the following code to ensure that this version or a later version is in use:

Variable version
PulseGetVersion version

```
if (version < N)
  | Handle the error
endif
```

8.4.13 PulseGroupGetComment

PulseGroupGetComment returns the comment for the current group.

Syntax

PulseGroupGetComment *comment, status*

comment A String variable to receive the group comment.

status A variable to receive the status of the operation.

Example

Variable status
String comment

PulseGroupGetComment comment, status

Discussion

The group comment is entered by the user, and may be null.

To set the current group, use PulseGroupSet.
☞ *PulseGroupSet*, p. 261.

8.4.14 PulseGroupGetCount

PulseGroupGetCount returns the number of groups in the data set.

Syntax

PulseGroupGetCount *count, status*

count A variable to receive the number of groups in the current data set.

status A variable to receive the status of the operation.

Example

Variable status
Variable group_count

PulseGroupGetCount count, status

Discussion

You must have a data set open.

8.4.15 PulseGroupGetExperiment

PulseGroupGetExperiment returns the group experiment number.

Syntax

PulseGroupGetExperiment *experiment, status*

experiment A variable to receive the group experiment number.

status A variable to receive the status of the operation.

Example

Variable status
Variable group_experiment

PulseGroupGetExperiment group_experiment, status

Discussion

The current group must be set. To set the current group, use PulseGroupSet. [↗ PulseGroupSet, p. 261.](#)

8.4.16 PulseGroupGetLabel

PulseGroupGetLabel returns the label for the current group.

Syntax

PulseGroupGetLabel *label, status*

label A String variable to receive the group label.

status A variable to receive the status of the operation.

Example

Variable status
String group_label

PulseGroupGetLabel group_label, status

Discussion

To set the current group, use PulseGroupSet. [↗ PulseGroupSet, p. 261.](#)

8.4.17 PulseGroupSet

PulseGroupSet selects the current group.

Syntax

PulseGroupSet *group, status*

group A value representing the group to select.

status A variable to receive the status of the operation.

Example

The following example sets the current group to the last group in a data set.

Variable status
Variable group_count

PulseGroupGetCount group_count, status

PulseGroupSet group_count, status

Discussion

You must have a data set open.

The groups in a data set are numbered from 1 to N, where N is the value returned by PulseGroupGetCount. [↗ PulseGroupGetCount, p. 259.](#)

8.4.18 PulseSegmentGetCount

PulseSegmentGetCount returns the number of segments in the current series.

Syntax

PulseSegmentGetCount *count, status*

count A variable to receive the number of stimulus segments in the current series.

status A variable to receive the status of the operation.

Example

Variable status
Variable segment_count

PulseSegmentGetCount count, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.19 PulseSegmentGetParameters

PulseSegmentGetParameters returns the parameters of the specified segment for the current sweep.

Syntax

```
PulseSegmentGetParameters
    segment, type, voltage, duration, status
```

segment A value representing the segment to use.

type A variable to receive the type of the segment.

voltage A variable to receive the voltage of the segment.

duration A variable to receive the duration of the segment. The duration is measured in samples.

status A variable to receive the status of the operation.

Example

The following example obtains the parameters of the last segment of the current series.

```
Variable status
Variable segment
Variable segment_type
Variable voltage
Variable duration
```

```
PulseSegmentGetCount segment, status
```

```
if (segment < 1)
    | Handle the error
endif
```

```
PulseSegmentGetParameters segment, segment_type,
    voltage, duration, status
```

Discussion

The segment type value is interpreted according to table 122.

Table 122 Segment types

Type	Interpretation
2	Conditioning
3	Continuous
4	Sine wave
5	Square wave

If the segment type is constant, conditioning, or continuous, the voltage represents a fixed output value.

If the segment type is ramp, the voltage represents the final voltage of the segment. The initial value is the final voltage of the previous segment.

If the segment type is either sine or square wave, the output is a constant value of specified voltage plus a sine wave or square wave waveform. The parameters of the waveform may be obtained using PulseSegmentGetWaveform. [PulseSegmentGetWaveform, p. 264.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [PulseSweepSet, p. 285.](#)

Segment numbers are in the range 1 to N, where N is the value returned by PulseSegmentGetCount. [PulseSegmentGetCount, p. 261.](#)

Table 122 Segment types

Type	Interpretation
0	Constant
1	Ramp

8.4.20 PulseSegmentGetRelevant

PulseSegmentGetRelevant returns the relevant x and y segments for the current series.

Syntax

PulseSegmentGetRelevant *x_segment, y_segment, status*

x_segment A variable to receive the series relevant x segment.

y_segment A variable to receive the series relevant y segment.

status A variable to receive the status of the operation.

Example

Variable status
Variable relevant_x
Variable relevant_y

```
PulseSegmentGetRelevant relevant_x, relevant_y,
status
```

Discussion

The relevant x segment is the stimulus segment containing the relevant independent variable. The relevant y segment is the acquired data segment containing the “interesting” data.

The returned segment numbers will be in the range 1 to N, where N is the value returned by PulseSegmentGetCount. [☞ PulseSegmentGetCount, p. 261.](#)

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.21 PulseSegmentGetTrigger

PulseSegmentGetTrigger returns the input data trigger for the series. The recorded data begins at the specified segment and sample within the stimulus record.

Syntax

PulseSegmentGetTrigger *segment, sample, status*

segment A variable to receive the stimulus segment at which recording begins.

sample A variable to receive the sample at which recording begins within the specified segment.

status A variable to receive the status of the operation.

Example

Variable status
Variable segment
Variable sample

```
PulseSegmentGetTrigger segment, sample, status
```

Discussion

You must have a series selected. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

Segment numbers are in the range 1 to N, where N is the value returned by PulseSegmentGetCount. [☞ PulseSegmentGetCount, p. 261.](#)

8.4.22 PulseSegmentGetWaveform

PulseSegmentGetWaveform returns the periodic waveform parameters for a series.

Syntax

PulseSegmentGetWaveform *period, voltage, status*

period A variable to receive the waveform period, measured in seconds.

voltage A variable to receive the waveform voltage, measured in volts.

status A variable to receive the status of the operation.

Example

Variable status
Variable period
Variable voltage

PulseSegmentGetWaveform period, voltage, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

The waveform parameters apply if the segment type is sine wave or square wave. You can use PulseSegmentGetParameters to obtain the type of a segment. [☞ PulseSegmentGetParameters, p. 262.](#)

The waveform parameters are only supplied for each series, not for each waveform stimulus segment.

8.4.23 PulseSeriesGetBackgroundNoise

PulseSeriesGetBackgroundNoise returns the background noise for the current series.

Syntax

PulseSeriesGetBackgroundNoise *noise, status*

noise A variable to receive the background noise.

status A variable to receive the status of the operation.

Example

Variable status
Variable noise

PulseSeriesGetBackgroundNoise noise, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.24 PulseSeriesGetBandwidth

PulseSeriesGetBandwidth returns the bandwidth for the current series.

Syntax

PulseSeriesGetBandwidth *bandwidth, status*

bandwidth A variable to receive the bandwidth.

status A variable to receive the status of the operation.

Example

Variable status
Variable bandwidth

PulseSeriesGetBandwidth bandwidth, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.25 PulseSeriesGetCellPotential

PulseSeriesGetCellPotential returns the cell potential for the current series.

Syntax

PulseSeriesGetCellPotential *cell_potential, status*

cell_potential A variable to receive the cell potential.

status A variable to receive the status of the operation.

Example

Variable status
Variable cell_potential

PulseSeriesGetCellPotential cell_potential, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.26 PulseSeriesGetComment

PulseSeriesGetComment returns the comment for the current series.

Syntax

PulseSeriesGetComment *comment, status*

comment A String variable to receive the series comment.

status A variable to receive the status of the operation.

Example

Variable status
String comment

PulseSeriesGetComment comment, status

Discussion

The series comment is entered by the user, and may be null.

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.27 PulseSeriesGetCount

PulseSeriesGetCount returns the number of series in the current group.

Syntax

PulseSeriesGetCount *count, status*

count A variable to receive the number of series in the current group.

status A variable to receive the status of the operation.

Example

Variable status
Variable series_count

PulseSeriesGetCount count, status

Discussion

The current group must be set. To set the current group, use PulseGroupSet. [↗ PulseGroupSet, p. 261.](#)

8.4.28 PulseSeriesGetHoldingPotential

PulseSeriesGetHoldingPotential returns the holding potential for the current series.

Syntax

PulseSeriesGetHoldingPotential *holding_potential, status*

holding_potential A variable to receive the holding potential.

status A variable to receive the status of the operation.

Example

Variable status

Variable holding_potential

```
PulseSeriesGetHoldingPotential holding_potential,
status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.29 PulseSeriesGetLabel

PulseSeriesGetLabel returns the label for the current series.

Syntax

PulseSeriesGetLabel *label, status*

label A String variable to receive the series label.

status A variable to receive the status of the operation.

Example

Variable status

String label

```
PulseSeriesGetLabel label, status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.30 PulseSeriesGetLeak

PulseSeriesGetLeak returns the leak parameters for the current series.

Syntax

```
PulseSeriesGetLeak count, scaling, potential,
alternate, alternate_averaging, delay, status
```

count A variable to receive the number of sweeps averaged for a leak trace.

scaling A variable to receive the scaling applied to the leak trace.

potential A variable to receive the leak holding potential.

alternate A variable to receive the leak alternate protocol.

alternate_averaging A variable to receive the leak alternate averaging protocol.

delay A variable to receive the delay between the leak and the stimulus.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable count
Variable scaling
Variable potential
Variable alternate
Variable alternate_averaging
Variable delay
```

```
PulseSeriesGetLeak count, scaling, potential, alternate,
alternate_averaging, delay, status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

If alternate is non-zero every second leak pulse is inverted relative to the leak holding potential.

If alternate averaging is non-zero every second leak template is inverted relative to the leak holding potential when

averaging over multiple traces.

The delay is the time between the leak pulses and the main stimulus pulse. When delay is positive, the leak pulses follow the main stimulus, otherwise the leak pulses precede the main stimulus.

8.4.31 PulseSeriesGetPipette

PulseSeriesGetPipette returns the leak parameters for the current series.

Syntax

PulseSeriesGetPipette *potential, resistance, pressure, status*

potential A variable to receive the pipette potential.

resistance A variable to receive the pipette resistance.

pressure A variable to receive the pipette pressure.

status A variable to receive the status of the operation.

Example

Variable status
Variable count
Variable potential
Variable resistance
Variable pressure

```
PulseSeriesGetPipette potential, resistance, pressure,
status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.32 PulseSeriesGetPotentialType

PulseSeriesGetPotentialType returns the stimulus potential type for the current series.

Syntax

PulseSeriesGetPotentialType *type, status*

type A variable to receive the potential type.

status A variable to receive the status of the operation.

Example

Variable status
Variable potential_type

```
PulseSeriesGetPotentialType potential_type, status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

The potential type is one of the values listed in table 123.

Table 123 Potential types

Typ e	Potential
0	Absolute stimulus
1	Relative stimulus
2	Absolute voltage

8.4.33 PulseSeriesGetRecordingMode

PulseSeriesGetRecordingMode returns the recording mode for the current series.

Syntax

PulseSeriesGetRecordingMode *mode, status*

mode A variable to receive the recording mode.

status A variable to receive the status of the operation.

Example

Variable status
Variable mode

PulseSeriesGetRecordingMode mode, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

The recording mode is one of the values listed in table 124.

Table 124 Recording modes

Mode	Protocol
0	In out
1	On cell
2	Out out
3	Whole cell
4	C clamp
5	V clamp

8.4.34 PulseSeriesGetSampleInterval

PulseSeriesGetSampleInterval returns the sampling interval in seconds for the current series.

Syntax

PulseSeriesGetSampleInterval *interval, status*

interval A variable to receive the sampling interval.

status A variable to receive the status of the operation.

Example

Variable status
Variable interval

PulseSeriesGetSampleInterval interval, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

All channels are sampled at the same interval. The interval measures the time between samples on the same channel.

8.4.35 PulseSeriesGetSealResistance

PulseSeriesGetSealResistance returns the seal resistance for the current series.

Syntax

PulseSeriesGetSealResistance *resistance, status*

resistance A variable to receive the seal resistance.

status A variable to receive the status of the operation.

Example

Variable status
Variable resistance

PulseSeriesGetSealResistance resistance, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.36 PulseSeriesGetSolutions

PulseSeriesGetSolutions returns the solution numbers for the current series.

Syntax

PulseSeriesGetSolutions *internal, external, status*

internal A variable to receive the internal solution number.

external A variable to receive the external solution number.

status A variable to receive the status of the operation.

Example

Variable status
Variable internal
Variable external

PulseSeriesGetSolutions internal, external, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.37 PulseSeriesGetStart

PulseSeriesGetStart returns the start time in seconds for the current series.

Syntax

PulseSeriesGetStart *start, status*

start A variable to receive the start time.

status A variable to receive the status of the operation.

Example

Variable status
Variable series_start

PulseSeriesGetStart series_start, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

The start time of a series is measured in seconds since the beginning of the experiment.

8.4.38 PulseSeriesGetTemperature

PulseSeriesGetTemperature returns the temperature for the current series.

Syntax

PulseSeriesGetTemperature *temperature, status*

temperature A variable to receive the **temperature**.

status A variable to receive the status of the operation.

Example

Variable status
Variable **temperature**

PulseSeriesGetTemperature **temperature**, status

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

8.4.39 PulseSeriesGetTime

PulseSeriesGetTime returns the date/time stamp for the start of the current series.

Syntax

PulseSeriesGetTime *series_time, status*

series_time A variable to receive the time stamp.

status A variable to receive the status of the operation.

Example

Variable status
Variable series_time

```
PulseSeriesGetTime series_time, status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

The time stamp is a numeric representation of the date and time for the series. It can be displayed in a table by setting the column format to be “data and time”.

8.4.40 PulseSeriesGetUserParameter1

PulseSeriesGetUserParameter1 returns the first user parameter for the current series.

Syntax

PulseSeriesGetUserParameter1 *session, name., value, units, status*

session A Long value representing the current session.

name A String variable to receive the parameter name.

value A Double variable to receive the parameter value.

units A String variable to receive the parameter units.

status A variable to receive the status of the operation.

Example

Variable status
String name
Variable parameter
String units

```
PulseSeriesGetUserParameter1 session, name,  
parameter, units, status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.41 PulseSeriesGetUserParameter2

PulseSeriesGetUserParameter2 returns the second user parameter for the current series.

Syntax

```
PulseSeriesGetUserParameter2 session, name., value,
units, status
```

session A Long value representing the current session.

name A String variable to receive the **parameter name**.

value A Double variable to receive the parameter value.

units A String variable to receive the parameter units.

status A variable to receive the status of the operation.

Example

```
Variable status
String name
Variable parameter
String units
```

```
PulseSeriesGetUserParameter2 session, name,
parameter, units, status
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.42 PulseSeriesSet

PulseSeriesSet selects the current series.

Syntax

```
PulseSeriesSet series, status
```

series The number of the series to use within the current group.

status A variable to receive the status of the operation.

Example

The following example sets the current series to the last series in the current group.

```
Variable status
Variable series
```

```
PulseSeriesGetCount series, status
```

```
if (series < 1)
    | Handle the error
endif
```

```
PulseSeriesSet series, status
```

Discussion

Series are numbered 1 to N, where N is the number of series in the current group. The number of series in the current group is returned by PulseSeriesGetCount. [☞ PulseSeriesGetCount, p. 266.](#)

8.4.43 PulseStimulusGetDAC

PulseStimulusGetDAC obtains the physical D/A channel.

Syntax

PulseStimulusGetDAC *dac, status*

dac A variable to receive the D/A channel number.

status A variable to receive the status of the operation.

Example

Variable status

Variable dac

```
PulseStimulusGetDAC dac, status
```

Discussion

You must have a data set open.

The physical channel number is the channel label from the data acquisition device. The channel number is one of the values shown in table 125.

Table 125 Stimulus DAC

DAC	Output channel
0	DAC 0
1	DAC 1
2	DAC 2
3	DAC 3
5	Default

If the dac value is 'Default', then the channel is the default 'V-membrane Out' channel.

8.4.44 PulseStimulusGetRange

PulseStimulusGetRange returns the range of the stimulus.

Syntax

PulseStimulusGetRange *minimum, maximum, status*

minimum A variable to receive the lower limit.

maximum A variable to receive the upper limit.

status A variable to receive the status of the operation.

Example

Variable status

Variable minimum

Variable maximum

```
PulseStimulusGetRange minimum, maximum, status
```

Discussion

You must have a data set open.

PulseStimulusGetRange returns the minimum and maximum possible values for the stimulus.

8.4.45 PulseStimulusGetTrigger

PulseStimulusGetTrigger returns the parameters for the specified trigger pulse.

Syntax

PulseStimulusGetTrigger *trigger, segment, time, duration, value, channel, status*

trigger A value specifying the trigger.

segment A variable to receive the trigger segment.

time A variable to receive the trigger time.

duration A variable to receive the trigger duration.

value A variable to receive the trigger value.

channel A variable to receive the trigger output channel.

status A variable to receive the status of the operation.

Example

Variable status
Variable segment
Variable time
Variable duration
Variable value
Variable channel

PulseStimulusGetTrigger 1, segment, time,
duration, value, channel, status

Discussion

The triggers are numbered 1 to 3. To determine how many triggers are available use PulseStimulusGetTriggerCount. [☞ PulseStimulusGetTriggerCount, p. 277.](#)

Time is the time of the trigger in seconds from the beginning of the segment. Duration is the duration of the trigger in seconds.

Channel can be one of the values shown in table 126.

Table 126 Trigger channels

Channel	Output channel
0	DAC 0
1	DAC 1
2	DAC 2
3	DAC 3
4	Digital
5	Default
6	None

If the channel is 'Default', then the channel is the default 'Trigger Out' channel. If the channel is 'None', then there is no trigger output.

For an analog channel, value is the output voltage. For a digital trigger, the lowest byte of value is the digital mask of the channels used. There are eight digital lines.

The first trigger, if enabled, determines when data recording starts. [☞ PulseSegmentGetTrigger, p. 263.](#)

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.46 PulseStimulusGetTriggerCount

PulseStimulusGetTriggerCount returns the number of triggers for the stimulus.

Syntax

PulseStimulusGetTriggerCount *count, status*

count A variable to receive the number of triggers for the stimulus.

status A variable to receive the status of the operation.

Example

Variable status
Variable trigger_count

PulseStimulusGetTriggerCount trigger_count, status

Discussion

The trigger count can range between 0 and 3. To get the parameters for a trigger use PulseStimulusGetTrigger. [☞ PulseStimulusGetTrigger, p. 276.](#)

You must have a data set open and the current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

8.4.47 PulseStimulusGetUnits

PulseStimulusGetUnits returns the units associated with the stimulus.

Syntax

PulseStimulusGetUnits *cunits, status*

units A String variable to contain the units.

status A variable to receive the status of the operation.

Example

Variable status
String units

PulseStimulusGetUnits units, status

Discussion

You must have a data set open.

8.4.48 PulseSweepGetAmplifier

PulseSweepGetAmplifier returns the amplifier settings for the current sweep.

Syntax

```
PulseSweepGetAmplifier gain_0, gain_1,
    capacitance, conductance, resistance, status
```

gain_0 A value to receive the gain for the first channel.

gain_1 A value to receive the gain for the second channel.

capacitance A value to receive the capacitance.

conductance A value to receive the series conductance.

resistance A value to receive the series resistance compensation.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable gain_0
Variable gain_1
Variable capacitance
Variable conductance
Variable resistance
```

```
PulseSweepGetAmplifier gain_0, gain_1, capacitance,
    conductance, resistance, status
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 285.](#)

The capacitance is the 'C Slow' capacitance in Farads.

The conductance is the 'G Series' conductance in Siemens.

The series resistance, 'R Series', is the inverse of the series conductance.

The series resistance compensation is the 'Rs Value' in Ohms.

8.4.49 PulseSweepGetCount

PulseSweepGetCount returns the number of sweeps in the current series.

Syntax

```
PulseSweepGetCount count, status
```

count A variable to receive the number of sweeps recorded in the current series.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable sweeps

PulseSweepGetCount sweeps, status
```

Discussion

You must have a series selected. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 274.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in the series.

8.4.50 PulseSweepGetHoldingPotential

PulseSweepGetHoldingPotential returns the holding potential for the current sweep.

Syntax

PulseSweepGetHoldingPotential *potential, status*

potential A variable to the sweep holding potential.

status A variable to receive the status of the operation.

Example

Variable status
Variable potential

PulseSweepGetHoldingPotential potential, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 285.](#)

8.4.51 PulseSweepGetLabel

PulseSweepGetLabel returns the label for the current sweep.

Syntax

PulseSweepGetLabel *label, status*

label A String variable to receive the sweep label.

status A variable to receive the status of the operation.

Example

Variable status
String sweep_label

PulseSweepGetLabel sweep_label, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 285.](#)

8.4.52 PulseSweepGetLeakAvailable

PulseSweepGetLeakAvailable returns a flag specifying whether or not leak data is available for the sweep.

Syntax

PulseSweepGetLeakAvailable *available, status*

available A Variable to receive the leak available flag. The flag is zero if no leak data is available for the sweep, non-zero if leak data is available for the sweep.

status A variable to receive the status of the operation.

Example

```
Variable status
Variable leak_available

PulseSweepGetLeakAvailable leak_available, status

if (leak_available != 0)
  | Process the leak data
endif
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 285.](#)

8.4.53 PulseSweepGetOnlineResult

PulseSweepGetOnlineResult returns an online result for the current sweep.

Syntax

PulseSweepGetOnlineResult *index, x_result, y_result*

index A value indicating the result to get.

x_result A variable to receive the x result.

y_result A variable to receive the y result.

Example

```
Variable x_result
Variable y_result

PulseSweepGetOnlineResult 0, x_result,
y_result
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 285.](#)

The index can be 0 or 1.

8.4.54 PulseSweepGetSampleCount

PulseSweepGetSampleCount returns the number of samples in the current sweep.

Syntax

PulseSweepGetSampleCount *samples, status*

samples A variable to receive the number of samples recorded in the current sweep.

status A variable to receive the status of the operation.

Example

Variable status
Variable samples

PulseSweepGetSampleCount samples, status

Discussion

The number of samples returned by PulseSweepGetSampleCount is the number of samples recorded for each channel.

The number of samples recorded may be less than the total duration of all stimulus segments if recording does not begin at the beginning of the first stimulus segment. [☞ PulseSegmentGetTrigger, p. 263.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 285.](#)

8.4.55 PulseSweepGetStart

PulseSweepGetStart returns the sweep start time.

Syntax

PulseSweepGetStart *sweep_time, data_time, status*

sweep_time A variable to receive the starting time of the sweep.

data_time A variable to receive the starting time of the sweep data.

status A variable to receive the status of the operation.

Example

Variable status
Variable sweep_time
Variable data_time

PulseSweepGetStart sweep_time, data_time, status

Discussion

The sweep time is measured in seconds since the beginning of the series. This is the time the stimulus starts.

The data time is measured in seconds since the beginning of the series. This is the starting time of recorded data. [☞ PulseSegmentGetTrigger, p. 263.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 285.](#)

8.4.56 PulseSweepGetTimer

PulseSweepGetTimer returns the timer value at the start of the sweep.

Syntax

PulseSweepGetTimer *timer, status*

timer A variable to receive the timer value at the start of the sweep.

status A variable to receive the status of the operation.

Example

Variable timer
Variable status

PulseSweepGetTimertimer, status

Discussion

The timer value is measured in seconds since the most recent timer reset.

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 285.](#)

8.4.57 PulseSweepGetUserParameter1

PulseSweepGetUserParameter1 returns the first user parameter for the current series.

Syntax

PulseSweepGetUserParameter1 *session, value, status*

session A Long value representing the current session.

value A Double variable to receive the parameter value.

status A variable to receive the status of the operation.

Example

Variable status
Variable value

PulseSweepGetUserParameter1 session, value, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 285.](#)

To get the name and units for the parameter, use PulseSeriesGetUserParameter1.

[↗ PulseSeriesGetUserParameter1, p. 273.](#)

8.4.58 PulseSweepGetUserParameter2

PulseSweepGetUserParameter2 returns the first user parameter for the current series.

Syntax

PulseSweepGetUserParameter2 *session, value, status*

session A Long value representing the current session.

value A Double variable to receive the parameter value.

status A variable to receive the status of the operation.

Example

Variable status

Variable value

PulseSweepGetUserParameter2 session, value, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 285.](#)

To get the name and units for the parameter, use PulseSeriesGetUserParameter2.

[☞ PulseSeriesGetUserParameter2, p. 274.](#)

8.4.59 PulseSweepRead

PulseSweepRead reads the data for one channel of the current sweep into a wave.

Syntax

PulseSweepRead *channel, start, samples, data, status*

channel The channel number containing the data to read.

start The number of the first sample within the sweep to read.

samples The number of samples to read.

data A wave to receive the data. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status

Variable samples

PulseSweepGetSampleCount samples, status

Make/D/N=(samples) data

PulseSweepRead channel, 0, samples, data, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 285.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [☞ PulseChannelGetCount, p. 253.](#)

The number of samples in the current sweep is provided by PulseSweepGetSampleCount. [☞ PulseSweepGetSampleCount, p. 281.](#)

8.4.60 PulseSweepReadLeak

PulseSweepReadLeak reads the leak template for the current sweep into a wave.

Syntax

PulseSweepReadLeak *start, samples, data, status*

start The number of the first sample within the sweep to read.

samples The number of samples to read.

data A wave to receive the data. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status

Variable samples

PulseSweepGetSampleCount samples, status

Make/D/N=(samples) leak_data

PulseSweepReadLeak 0, samples, leak_data, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 285.](#)

The number of samples in the current sweep is provided by PulseSweepGetSampleCount. [☞ PulseSweepGetSampleCount, p. 281.](#)

8.4.61 PulseSweepReadRaw

PulseSweepReadRaw reads the raw data without the leak correction for the first channel of the current sweep into a wave.

Syntax

PulseSweepReadRaw *start, samples, data, status*

start The number of the first sample within the sweep to read.

samples The number of samples to read.

data A wave to receive the data. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status

Variable samples

PulseSweepGetSampleCount samples, status

Make/D/N=(samples) data

PulseSweepReadRaw 0, samples, data, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 285.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [☞ PulseChannelGetCount, p. 253.](#)

The number of samples in the current sweep is provided by PulseSweepGetSampleCount. [☞ PulseSweepGetSampleCount, p. 281.](#)

8.4.62 PulseSweepReadStimulus

PulseSweepReadStimulus reads the output data for the current sweep into a wave.

Syntax

PulseSweepReadStimulus *start, samples, stimulus, status*

start The number of the first sample within the sweep to read.

samples The number of samples to read.

stimulus A wave to receive the stimulus. The wave must be long enough to contain the specified number of samples.

status A variable to receive the status of the operation.

Example

Variable status

Variable samples

PulseSweepGetSampleCount samples, status

Make/D/N=(samples) data

PulseSweepReadStimulus 0, samples, data, status

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 285.](#)

The number of samples in the current sweep is provided by PulseSweepGetSampleCount. That is, the returned stimulus samples are those that correspond to the acquired data returned by PulseSweepRead. [☞ PulseSweepRead, p. 283.](#)
[☞ PulseSweepGetSampleCount, p. 281.](#)

8.4.63 PulseSweepSet

PulseSweepSet selects the current sweep.

Syntax

PulseSweepSet *sweep, status*

sweep A value representing the sweep to set.

status A variable to receive the status of the operation.

Example

This example sets the current sweep to the last sweep in the current series.

Variable status

Variable sweeps

PulseSweepGetCount sweeps, status

if (sweeps < 1)

 | *Handle the error*

endif

PulseSweepSet sweeps, status

Discussion

Sweeps are numbered 1 to N, where N is the number of sweeps in the current series. The number of sweeps in the current series is provided by PulseSweepGetCount. [☞ PulseSweepGetCount, p. 278.](#)

You must have a series selected. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 274.](#)

9 Pulse: Visual Basic

Section	
9.1 Using DataAccess	p. 286
9.2 Getting Started	p. 286
9.3 Robust Processing	p. 289
9.4 Operations	p. 290
9.5 Reference	p. 292

9.1 Using DataAccess

To make DataAccess available to a Visual Basic program, add the file PulseVB.bas to the project. Ensure that the file PulseVB.dll is in your path.

To make DataAccess available to an Systat Software SigmaPlot script module, include the following statement in the module:

```
'#Uses "PulseVB.bas"
```

The leading quote (!) is required. The bas file must be in the same folder as your SigmaPlot project, or you must explicitly specify the path to the bas file.

9.1.1 Status

Each operation returns a status code. This code is zero if no error is detected.

If an error is detected, the returned status value is non-zero. It can be translated to a text string using PulseGetStatusText.

9.1.2 Data Types

Integer values are passed as the data type "Long". Real values are passed as the data type "Double". Arrays are passed by passing the first element in the array. Character string values are passed as the data type "Variant". The data type "String" is not used because when Visual Basic passes parameters to dlls, it performs undesired translations from Unicode to ASCII.

9.1.3 Values

All duration values are measured in seconds.

All amplitude values are measured in volts, taking into account any scaling specified in the file.

9.1.4 Calls

Almost all DataAccess procedures return a value. In Visual Basic, you have several choices of syntax when you call such procedures. For example, you can call PulseChannelGetCount as follows:

- 1 As a function, using the return value:

```
result = PulseChannelGetCount(session, count)
```

- 2 As a subroutine using "Call":

```
Call PulseChannelGetCount(session, count)
```

- 3 Directly as a command. In this case, the parameters are not enclosed in parentheses:

```
PulseChannelGetCount session, count
```

Any of the methods work. The examples in this manual generally use the command form.

9.2 Getting Started

This section explains how to access an Pulse data set. It

contains a step by step description of the operations to perform to open a data set and read the data in it.

9.2.1 Opening a Session

Open a session using the PulseSessionOpen operation. [☞ PulseSessionOpen, p. 315.](#)

PulseSessionOpen returns a handle. You must pass this handle to other operations that use that session.

9.2.2 Opening a Data Set

Open an Pulse data set using the PulseFileOpen operation. [☞ PulseFileOpen, p. 297.](#)

In order to open a data set, you must supply the path to the DAT file.

The following example opens a session, then opens a data set with the path “f:\test.dat”. After the data set is open, it closes the data set and the session.

```
Dim session As Long
Dim status As Long
Dim file_name As Variant
file_name = "f:\test.dat"

PulseSessionOpen session
status = PulseFileOpen(session, file_name)

If status <> 0 Then
    ' The data set could not be opened. Handle the
    ' error and do not call PulseFileClose
End If

PulseFileClose session
PulseSessionClose session
```

9.2.3 File Parameters

Once you have opened a data set, you can determine the parameters of the data set. The parameters include the creation date and time and the data set comment. [☞ File Parameter Operations, p. 290.](#)

The following example obtains data set creation date and

time:

```
Dim file_date_time As Date

PulseFileGetTime session, file_date_time
```

9.2.4 Groups

A data set is a sequence of groups. You use the group operations to select a group and determine its parameters. [☞ Group Operations, p. 290.](#)

You use PulseGroupGetCount to determine the number of groups in a data set. [☞ PulseGroupGetCount, p. 299.](#)

Groups are numbered 1 to N, where N is the number of groups in a data set. You use PulseGroupSet to set the group to use. [☞ PulseGroupSet, p. 301.](#)

The following example obtains the label of each group in a data set. The label is a text string.

```
Dim group_count As Long
Dim current_group As Long
Dim group_label As Variant

PulseGroupGetCount session, group_count

If group_count > 0 Then
    For current_group = 1 To group_count
        PulseGroupSet session, current_group
        PulseGroupGetLabel session, group_label
        ' Process the label here
    Next current_group
Else
    ' The data set contains no groups
End If
```

A file may have no groups if no data was acquired.

9.2.5 Series

A group is a sequence of series. You use the series operations to select a series and determine its parameters. [☞ Series Operations, p. 291.](#)

You use PulseSeriesGetCount to determine the number of series in a group. [☞ PulseSeriesGetCount, p. 306.](#)

Series are numbered 1 to N, where N is the number of

series in a group. You use `PulseSeriesSet` to set the series to use. [☞ PulseSeriesSet, p. 314.](#)

The following example obtains the label of each series in the current group. The label is a text string.

```
Dim series_count As Long
Dim current_series As Long
Dim series_label As Variant

PulseSeriesGetCount session, series_count

If series_count > 0 Then
  For current_series = 1 To series_count
    PulseSeriesSet session, current_series
    PulseSeriesGetLabel session, series_label
    ' Process label here
  Next current_series
Else
  ' The group contains no series
End If
```

A group may have no series if no data was acquired.

9.2.6 Sweeps

A series is a sequence of sweeps. You use the sweep operations to select a sweep and determine its parameters. [☞ Sweep Operations, p. 291.](#)

You use `PulseSweepGetCount` to determine the number of sweeps in a sweep. [☞ PulseSweepGetCount, p. 319.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in a series. You use `PulseSweepSet` to set the sweep to use. [☞ PulseSweepSet, p. 326.](#)

The following example obtains the number of samples in each sweep in the current series.

```
Dim sweep_count As Long
Dim current_sweep As Long
Dim sweep_samples As Long

PulseSweepGetCount session, sweep_count

If sweep_count > 0 Then
  For current_sweep = 1 To sweep_count
    PulseSweepSet session, current_sweep
    PulseSweepGetSampleCount
      session, sweep_samples
```

```
' Process sample count here
Next current_sweep
Else
  ' The series contains no sweep
End If
```

A series may have no sweep if no data was acquired.

9.2.7 Segments

A stimulus template is described by a sequence of segments. Although all sweeps in a series use the same stimulus template, the duration and voltage of each segment can vary with each sweep.

Segment numbers range from 1 to N, where N is the number of segments in the series. You use `PulseSegmentGetCount` to determine the number of segments in a series. [☞ PulseSegmentGetCount, p. 301.](#)

9.2.8 Channels

`PulseChannelGetCount` returns the number of channels recorded in a sweep. Normally the number of channels recorded in each sweep in a series will be the same when a data set is acquired, but channels can be removed from individual sweeps during data set editing, leaving a series with varying numbers of channels recorded in each sweep. [☞ PulseChannelGetCount, p. 293.](#)

The following example obtains the number of channels recorded in the current sweep:

```
Dim channel_count As Long

PulseChannelGetCount session, channel_count
```

Channel numbers range from 0 to N-1, where N is the number of channels in the sweep. Existing versions of Pulse never record more than two channels in a sweep.

9.2.9 Reading Data

To read a sweep, use `PulseSweepRead`. `PulseSweepRead` can read either a whole sweep or sections of a sweep. [☞ PulseSweepRead, p. 324.](#)

Normally one would read whole sweeps but it may be necessary to read sections of an sweep if you have insufficient

memory available to read whole sweeps.

Reading Sweeps

The following example reads each sweep of a series into an array.

```
Dim sweep_count As Long
Dim current_sweep As Long
Dim samples As Long
Dim buffer() As Double

PulseSweepGetCount session, segment, sweep_count

If sweep_count > 0 Then
  For current_sweep = 1 To sweep_count
    PulseSweepSet session, current_sweep
    PulseSweepGetSampleCount session, samples
    ReDim buffer(samples-1)

    PulseSweepRead session, channel, 0, samples,
      buffer(0)
    ' Data processing goes here
  Next current_sweep
Else
  ' The series has no sweeps
End If
```

Sweeps and Channels

Each channel in an sweep has the same number of data samples.

The following example reads data from each channel of the current sweep into an array.

```
Dim channel_count As Long
Dim channel As Long
Dim samples As Long
Dim buffer() As Double

PulseSweepGetSampleCount session, samples

ReDim buffer(samples-1)

PulseChannelGetCount session, channel_count

For channel = 0 To channel_count-1
```

```
PulseSweepRead session, channel, 0, samples,
  buffer(0)
' Data processing goes here
Next channel
```

Reading Sections of a Sweep

The following example reads the first “length” values of the first sweep into a array.

```
Dim samples As Long
Dim buffer() As Double

PulseSweepGetSampleCount session, samples

If samples > length Then
  samples = length
End If

ReDim buffer(samples-1)

PulseSweepRead session, channel, 0, samples, buffer(0)
```

9.2.10 Closing a Data Set

To close a data set, use `PulseFileClose`. [☞ PulseFileClose, p. 294.](#)

The following example closes a Pulse data set.

```
PulseFileClose session
```

9.2.11 Closing a Session

To close a session, use `PulseSessionClose`. [☞ PulseSessionClose, p. 315.](#)

9.3 Robust Processing

If you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

All operations return a status value of type `Long`. The value is zero if the operation was successful, and non-zero if an error was detected.

If an error is detected, you can use `PulseGetStatusText` to

translate the status code to a message. [☞ PulseGetStatusText](#), p. 298.

The following example shows how the user might be allowed to select a file. The example does not terminate until the user selects a valid Pulse data set.

```
Dim filespec As Variant
Dim status As Long
Dim session As Long

PulseSessionOpen session

Do
    ' Add code here to obtain the file specification in "filespec"
    status = PulseFileOpen(session, filespec)

    If status = 0 Then
        Exit Do
    End If

    ' Translate the status code to a message here and
    ' display it for the user
Loop
```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

9.4 Operations

This chapter describes each of the operations provided by DataAccess. The operations are grouped by category.

9.4.1 Common Operations

The common operations are valid regardless of whether a data set is open or not. The common operations are shown in table 127.

Table 127 Common operations

Operation	
PulseGetVersion	p. 298
PulseGetStatusText	p. 298

9.4.2 Session Operations

The session operations allow you to open and close a session. The session operations are shown in

Table 128 Session operations

Operation	
PulseSessionOpen	p. 315
PulseSessionClose	p. 315

9.4.3 File Operations

The file operations allow you to access a Pulse data set. The file operations are shown in table 129.

Table 129 File operations

Operation	
PulseFileOpen	p. 297
PulseFileClose	p. 294

The file operations are valid when a session is open.

9.4.4 File Parameter Operations

The file parameter operations allow you to obtain data set parameters. The file parameter operations are shown in table 130.

Table 130 File parameter operations

Operation	
PulseFileGetComment	p. 295
PulseFileGetPulseVersion	p. 295
PulseFileGetSignature	p. 296
PulseFileGetTime	p. 296
PulseFileGetVersion	p. 297

The file parameter operations are valid when a data set is open.

9.4.5 Group Operations

The group operations allow you to access each group of a

data set. The group operations are shown in table 131.

Table 131 Group operations

Operation	
PulseGroupGetComment	p. 299
PulseGroupGetCount	p. 299
PulseGroupGetExperiment	p. 300
PulseGroupGetLabel	p. 300
PulseGroupSet	p. 301

The group operations are valid when a data set is open. Most require that a current group be set.

9.4.6 Series Operations

The series operations allow you to access each series of a group. The series operations are shown in table 132.

Table 132 Series operations

Operation	
PulseSeriesGetBackgroundNoise	p. 304
PulseSeriesGetBandwidth	p. 305
PulseSeriesGetCellPotential	p. 305
PulseSeriesGetComment	p. 306
PulseSeriesGetCount	p. 306
PulseSeriesGetHoldingPotential	p. 307
PulseSeriesGetLabel	p. 307
PulseSeriesGetLeak	p. 308
PulseSeriesGetPipette	p. 309
PulseSeriesGetPotentialType	p. 309
PulseSeriesGetRecordingMode	p. 310
PulseSeriesGetSampleInterval	p. 310
PulseSeriesGetSealResistance	p. 311
PulseSeriesGetSolutions	p. 311
PulseSeriesGetStart	p. 312
PulseSeriesGetTemperature	p. 312
PulseSeriesGetUserParameter1	p. 313
PulseSeriesGetUserParameter2	p. 314
PulseSeriesSet	p. 314

The series operations are valid when a data set is open. Most require that a current series be set.

9.4.7 Sweep Operations

The sweep operations allow you to access each sweep of a

series. The sweep operations are shown in table 133.

Table 133 Sweep operations

Operation	
PulseSweepGetAmplifier	p. 319
PulseSweepGetCount	p. 319
PulseSweepGetHoldingPotential	p. 320
PulseSweepGetLabel	p. 320
PulseSweepGetLeakAvailable	p. 321
PulseSweepGetSampleCount	p. 322
PulseSweepGetStart	p. 322
PulseSweepGetTimer	p. 323
PulseSweepGetUserParameter1	p. 323
PulseSweepGetUserParameter2	p. 324
PulseSweepRead	p. 324
PulseSweepReadLeak	p. 325
PulseSweepReadRaw	p. 325
PulseSweepReadStimulus	p. 326
PulseSweepSet	p. 326

The sweep operations are valid when a data set is open. Most require that a current sweep be set.

9.4.8 Channel Operations

The channel operations allow you to obtain information about the acquired data channels. The channel operations are shown in table 134.

Table 134 Channel operations

Operation	
PulseChannelGetCount	p. 293
PulseChannelGetADC	p. 292
PulseChannelGetRange	p. 293
PulseChannelGetUnits	p. 294

9.4.9 Stimulus Operations

The stimulus operations allow you to obtain information about the stimulus protocol. The stimulus operations are shown in table 135.

Table 135 Stimulus operations

Operation	
PulseStimulusGetDAC	p. 316
PulseStimulusGetRange	p. 316

Table 135 Stimulus operations

Operation	
PulseStimulusGetTrigger	p. 317
PulseStimulusGetTriggerCount	p. 318
PulseStimulusGetUnits	p. 318

9.4.10 Segment Operations

The segment operations allow you to access information about the stimulus segments. The segment operations are shown in table 136.

Table 136 Segment operations

Operation	
PulseSegmentGetCount	p. 301
PulseSegmentGetParameters	p. 302
PulseSegmentGetRelevant	p. 303
PulseSegmentGetWaveform	p. 304
PulseSegmentGetTrigger	p. 303

9.5 Reference

This section lists each DataAccess operation in alphabetical order.

9.5.1 PulseChannelGetADC

PulseChannelGetADC obtains the physical A/D channel.

Syntax

PulseChannelGetADC *session, channel, adc*

session A Long value representing the current session.

channel A Long value representing the channel.

adc An Long variable to receive the A/D channel number.

Example

```
Dim adc As Long
```

```
PulseChannelGetADC session, channel, adc
```

Discussion

The logical channel number may differ from the physical A/D channel used. The physical channel number is the one labeled on the data acquisition device.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. To determine the number of channels, use PulseChannelGetCount.

☞ *PulseChannelGetCount*, p. 293.

9.5.2 PulseChannelGetCount

PulseChannelGetCount returns the number of channels recorded in the current sweep.

Syntax

```
PulseChannelGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of channels recorded in the current sweep.

Example

```
Dim channels As Long
```

```
PulseChannelGetCount session, channels
```

Discussion

The number of channels recorded may vary from sweep to sweep within a series.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep.

To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

9.5.3 PulseChannelGetRange

PulseChannelGetRange returns the range of the specified channel.

Syntax

```
PulseChannelGetRange session, channel, minimum,
maximum
```

session A Long value representing the current session.

channel A Long value representing the channel to use.

minimum A Double variable to receive the lower limit.

maximum A Double variable to receive the upper limit.

Example

```
Dim minimum As Double
```

```
Dim maximum As Double
```

```
PulseChannelGetRange session, 0, minimum,
maximum
```

Discussion

PulseChannelGetRange returns the minimum and maximum possible values for the specified channel.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. To determine the number of channels, use PulseChannelGetCount. [☞ PulseChannelGetCount, p. 293.](#)

The current sweep must be set. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

9.5.4 PulseChannelGetUnits

PulseChannelGetUnits obtains the units associated with a channel.

Syntax

PulseChannelGetUnits *session, units*

session A Long value representing the current session.

channel A Long value representing the channel.

units A Variant variable to receive the units.

Example

Dim units As Variant

PulseChannelGetUnits session, channel, units

Discussion

The units are entered by the user.

Channels are numbered 0 to N-1, where N is the number of channels in the sweep. To determine the number of channels, use `PulseChannelGetCount`.
 ☞ *PulseChannelGetCount*, p. 293.

The current series must be set. The units are set independently for each series. To set the current series, use `PulseSeriesSet`. ☞ *PulseSeriesSet*, p. 314.

9.5.5 PulseFileClose

PulseFileClose terminates processing of the Pulse data set.

Syntax

PulseFileClose *session*

session A Long value representing the current session.

Example

PulseFileClose session

Discussion

You must have a data set open.

You should perform `PulseFileClose` as soon as you complete processing of a data set.

9.5.6 PulseFileGetComment

PulseFileGetComment returns the comment for the data set.

Syntax

```
PulseFileGetComment session, comment
```

session A Long value representing the current session.

comment A Variant variable in which the data set comment is returned.

Example

```
Dim text As Variant
```

```
PulseFileGetComment session, text
```

Discussion

You must have a data set open.

9.5.7 PulseFileGetPulseVersion

PulseFileGetPulseVersion returns the version of Pulse which created the data set.

Syntax

```
PulseFileGetPulseVersion session, version
```

session A Long value representing the current session.

version A Variant variable in which the Pulse version is returned.

Example

```
Dim version As Variant
```

```
PulseFileGetPulseVersion session, version
```

Discussion

You must have a file open.

9.5.8 PulseFileGetSignature

PulseFileGetSignature returns a unique signature for the data file.

Syntax

```
PulseFileGetSignature session, signature
```

session A Long value representing the current session.

signature A Variant variable in which the file signature is returned.

Example

```
Dim signature As Variant
```

```
PulseFileGetSignature session, signature
```

Discussion

You must have a file open.

The returned signature is a string containing a 32 character hexadecimal value.

The signature is calculated from information in the file. Except for a very unlikely coincidence, the signature should uniquely identify the file.

9.5.9 PulseFileGetTime

PulseFileGetTime returns the creation date and time of the data set.

Syntax

```
PulseFileGetTime session, date_time
```

session A Long value representing the current session.

date_time A Date variable to receive the data set creation date and time.

Example

```
Dim date_time As Date
```

```
PulseFileGetTime session, date_time
```

Discussion

You must have a data set open.

9.5.10 PulseFileGetVersion

PulseFileGetVersion returns the Pulse format version of the file.

Syntax

PulseFileGetVersion *session, version*

session A Long value representing the current session.

version A Variant variable in which the file version is returned.

Example

```
Dim version As Variant
```

```
PulseFileGetVersion session, version
```

Discussion

You must have a file open.

9.5.11 PulseFileOpen

PulseFileOpen prepares a Pulse data set for processing.

Syntax

PulseFileOpen *session, path*

session A Long value representing the current session.

path A Variant value representing the data set path.

Example

```
Dim file_name As Variant
file_name = "Data.dat"
PulseFileOpen session, file_name
```

Discussion

You must not have a data set open.

A Pulse data set consists of three files, with the extensions PUL, PGF, and DAT. All three files must be present in the same directory for PulseFileOpen to succeed.

Once a data set is open, you can access data set parameters. [☞ File Parameter Operations, p. 290](#). You can process the groups in the data set. [☞ Group Operations, p. 290](#).

When you are done with a data set, you should close it. [☞ PulseFileClose, p. 294](#).

PulseFileOpen invalidates the current group. You must set the current group using PulseGroupSet. [☞ PulseGroupSet, p. 301](#).

9.5.12 PulseGetStatusText

PulseGetStatusText translates a status value returned by a DataAccess operation to a text string.

Syntax

PulseGetStatusText *session, status, message*

session A Long value representing the current session.

status A Long value representing the status value to translate.

message A Variant variable in which the translated text is returned.

Example

```
Dim status As Long
Dim message As Variant
Dim file_name As Variant
file_name = "Sample.dat"

status = PulseFileOpen(session, file_name)

If status <> 0 Then
    PulseGetStatusText session, status, message
    ' Display the message here
End If
```

Discussion

PulseGetStatusText does not return a value, so it should not be called as a function.

You need not have a data set open.

PulseGetStatusText is the only means provided to interpret status values.

9.5.13 PulseGetVersion

PulseGetVersion returns the version number of the DataAccess package.

Syntax

PulseGetVersion *version*

version A Long variable to receive the DataAccess version number.

Example

```
Dim version As Long

PulseGetVersion version
```

Discussion

You need not have a data set open.

If you are writing a set of procedures using DataAccess, you can use PulseGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess. You can use the following code to ensure that this version or a later version is in use:

```
Dim version As Long
PulseGetVersion version

If version < N Then
    ' Handle the error
End If
```

9.5.14 PulseGroupGetComment

PulseGroupGetComment returns the comment for the current group.

Syntax

```
PulseGroupGetComment session, comment
```

session A Long value representing the current session.

comment A Variant variable in which the group comment is returned.

Example

```
Dim text As Variant
```

```
PulseGroupGetComment session, text
```

Discussion

The current group must be set. To set the current group, use PulseGroupSet. [↗ PulseGroupSet, p. 301.](#)

The group comment is entered by the user, and may be null.

9.5.15 PulseGroupGetCount

PulseGroupGetCount returns the number of groups in the data set.

Syntax

```
PulseGroupGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of groups in the current data set.

Example

```
Dim group_count As Long
```

```
PulseGroupGetCount session, count
```

Discussion

You must have a data set open.

9.5.16 PulseGroupGetExperiment

PulseGroupGetExperiment returns the group experiment number.

Syntax

PulseGroupGetExperiment *session, experiment*

session A Long value representing the current session.

experiment A Long variable to receive the group experiment number.

Example

```
Dim group_experiment As Long
PulseGroupGetExperiment session, group_experiment
```

Discussion

The current group must be set. To set the current group, use PulseGroupSet. [↗ PulseGroupSet, p. 301.](#)

9.5.17 PulseGroupGetLabel

PulseGroupGetLabel returns the label for the current group.

Syntax

PulseGroupGetLabel *session, label*

session A Long value representing the current session.

label A Variant variable to receive the group label

Example

```
Dim text As Variant
PulseGroupGetLabel session, text
```

Discussion

The current group must be set. To set the current group, use PulseGroupSet. [↗ PulseGroupSet, p. 301.](#)

9.5.18 PulseGroupSet

PulseGroupSet selects the current group.

Syntax

PulseGroupSet *session, group*

session A Long value representing the current session.

group A Long value representing the group to select.

Example

The following example sets the current group to the last group in a data set.

```
Dim group_count As Long
PulseGroupGetCount session, group_count
PulseGroupSet session, group_count
```

Discussion

You must have a data set open.

The groups in a data set are numbered from 1 to N, where N is the value returned by PulseGroupGetCount. [↗ PulseGroupGetCount, p. 299.](#)

9.5.19 PulseSegmentGetCount

PulseSegmentGetCount returns the number of segments in the current series.

Syntax

PulseSegmentGetCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of stimulus segments in the current series.

Example

```
Dim segment_count As Long
PulseSegmentGetCount session, count
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.20 PulseSegmentGetParameters

PulseSegmentGetParameters returns the parameters of the specified segment for the current sweep.

Syntax

```
PulseSegmentGetParameters
    session, segment, type, voltage, duration
```

session A Long value representing the current session.

segment A Long value representing the segment to use.

type A Long variable to receive the type of the segment.

voltage A Double variable to receive the voltage of the segment.

duration A Long variable to receive the duration of the segment. The duration is measured in samples.

Example

The following example obtains the parameters of the last segment of the current series.

```
Dim segment As Long
Dim segment_type As Long
Dim voltage As Double
Dim duration As Long

PulseSegmentGetCount session, segment

If segment < 1 Then
    ' Handle the error
End If

PulseSegmentGetParameters session, segment,
    segment_type, voltage, duration
```

Discussion

The segment type value is interpreted according to table 137.

Table 137 Segment types

Type	Interpretation
2	Conditioning
3	Continuous
4	Sine wave
5	Square wave

If the segment type is constant, conditioning, or continuous, the voltage represents a fixed output value.

If the segment type is ramp, the voltage represents the final voltage of the segment. The initial value is the final voltage of the previous segment.

If the segment type is either sine or square wave, the output is a constant value of specified voltage plus a sine wave or square wave waveform. The parameters of the waveform may be obtained using PulseSegmentGetWaveform. [PulseSegmentGetWaveform, p. 304.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [PulseSweepSet, p. 326.](#)

Segment numbers are in the range 1 to N, where N is the value returned by PulseSegmentGetCount. [PulseSegmentGetCount, p. 301.](#)

Table 137 Segment types

Type	Interpretation
0	Constant
1	Ramp

9.5.21 PulseSegmentGetRelevant

PulseSegmentGetRelevant returns the relevant x and y segments for the current series.

Syntax

```
PulseSegmentGetRelevant session, x_segment, y_segment
```

session A Long value representing the current session.

x_segment A Long variable to receive the series relevant x segment.

y_segment A Long variable to receive the series relevant y segment.

Example

```
Dim relevant_x As Long
Dim relevant_y As Long

PulseSegmentGetRelevant
  session, relevant_x, relevant_y
```

Discussion

The relevant x segment is the stimulus segment containing the relevant independent variable. The relevant y segment is the acquired data segment containing the “interesting” data.

The returned segment numbers will be in the range 1 to N, where N is the value returned by PulseSegmentGetCount. [☞ PulseSegmentGetCount, p. 301.](#)

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.22 PulseSegmentGetTrigger

PulseSegmentGetTrigger returns the input data trigger for the series. The recorded data begins at the specified segment and sample within the stimulus record.

Syntax

```
PulseSegmentGetTrigger session, segment, sample
```

session A Long value representing the current session.

segment A Long variable to receive the stimulus segment at which recording begins.

sample A Long variable to receive the sample at which recording begins within the specified segment.

Example

```
Dim segment As Long
Dim sample As Long

PulseSegmentGetTrigger session, segment, sample
```

Discussion

You must have a series selected. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

Segment numbers are in the range 1 to N, where N is the value returned by PulseSegmentGetCount. [☞ PulseSegmentGetCount, p. 301.](#)

9.5.23 PulseSegmentGetWaveform

PulseSegmentGetWaveform returns the periodic waveform parameters for a series.

Syntax

PulseSegmentGetWaveform *session, period, voltage*

session A Long value representing the current session.

period A Double variable to receive the waveform period, measured in seconds.

voltage A Double variable to receive the waveform voltage, measured in volts.

Example

```
Dim period As Double
Dim voltage As Double
```

```
PulseSegmentGetWaveform session, period, voltage
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

The waveform parameters apply if the segment type is sine wave or square wave. You can use PulseSegmentGetParameters to obtain the type of a segment. [☞ PulseSegmentGetParameters, p. 302.](#)

The waveform parameters are only supplied for each series, not for each waveform stimulus segment.

9.5.24 PulseSeriesGetBackgroundNoise

PulseSeriesGetBackgroundNoise returns the background noise for the current series.

Syntax

PulseSeriesGetBackgroundNoise *session, noise*

session A Long value representing the current session.

noise A Double variable to receive the background noise.

Example

```
Dim noise As Double
```

```
PulseSeriesGetBackgroundNoise session, noise
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.25 PulseSeriesGetBandwidth

PulseSeriesGetBandwidth returns the bandwidth for the current series.

Syntax

```
PulseSeriesGetBandwidth session, bandwidth
```

session A Long value representing the current session.

bandwidth A Double variable to receive the bandwidth.

Example

```
Dim bandwidth As Double
```

```
PulseSeriesGetBandwidth session, bandwidth
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.26 PulseSeriesGetCellPotential

PulseSeriesGetCellPotential returns the cell potential for the current series.

Syntax

```
PulseSeriesGetCellPotential session, cell_potential
```

session A Long value representing the current session.

cell_potential A Double variable to receive the cell potential.

Example

```
Dim cell_potential As Double
```

```
PulseSeriesGetCellPotential session, cell_potential
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.27 PulseSeriesGetComment

PulseSeriesGetComment returns the comment for the current series.

Syntax

```
PulseSeriesGetComment session, comment
```

session A Long value representing the current session.

comment A Variant variable in which the series comment is returned.

Example

```
Dim text As Variant
```

```
PulseSeriesGetComment session, text
```

Discussion

You must have a data set open.

The series comment is entered by the user, and may be null.

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.28 PulseSeriesGetCount

PulseSeriesGetCount returns the number of series in the current group.

Syntax

```
PulseSeriesGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of series in the current group.

Example

```
Dim series_count As Long
```

```
PulseSeriesGetCount session, count
```

Discussion

The current group must be set. To set the current group, use PulseGroupSet. [↗ PulseGroupSet, p. 301.](#)

9.5.29 PulseSeriesGetHoldingPotential

PulseSeriesGetHoldingPotential returns the holding potential for the current series.

Syntax

```
PulseSeriesGetHoldingPotential session, holding_potential
```

session A Long value representing the current session.

holding_potential A Double variable to receive the holding potential.

Example

```
Dim holding_potential As Double
```

```
PulseSeriesGetHoldingPotential session,  
holding_potential
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.30 PulseSeriesGetLabel

PulseSeriesGetLabel returns the label for the current series.

Syntax

```
PulseSeriesGetLabel session, label
```

session A Long value representing the current session.

label A Variant variable to receive the series label

Example

```
Dim text As Variant
```

```
PulseSeriesGetLabel session, text
```

Discussion

You must have a data set open.

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.31 PulseSeriesGetLeak

PulseSeriesGetLeak returns the leak parameters for the current series.

Syntax

PulseSeriesGetLeak *session, count, scaling, potential, alternate, alternate_averaging, delay*

session A Long value representing the current session.

count A Long variable to receive the number of sweeps averaged for a leak trace.

scaling A Double variable to receive the scaling applied to the leak trace.

potential A Double variable to receive the leak holding potential.

alternate A Long variable to receive the leak alternate protocol.

alternate_averaging A Long variable to receive the leak alternate averaging protocol.

delay A Double variable to receive the delay between the leak and the stimulus.

Example

```
Dim count As Long
Dim scaling As Double
Dim potential As Double
Dim alternate As Long
Dim alternate_averaging As Long
Dim delay As Double
```

```
PulseSeriesGetLeak session, count, scaling, potential,
alternate, alternate_averaging, delay
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

If alternate is non-zero every second leak pulse is inverted relative to the leak holding potential.

If alternate averaging is non-zero every second leak tem-

plate is inverted relative to the leak holding potential when averaging over multiple traces.

The delay is the time between the leak pulses and the main stimulus pulse. When delay is positive, the leak pulses follow the main stimulus, otherwise the leak pulses precede the main stimulus.

9.5.32 PulseSeriesGetPipette

PulseSeriesGetPipette returns the pipette parameters for the current series.

Syntax

PulseSeriesGetPipette *session, potential, resistance, pressure*

session A Long value representing the current session.

potential A Double variable to receive the pipette potential.

resistance A Double variable to receive the pipette resistance.

pressure A Double variable to receive the pipette pressure.

Example

```
Dim count As Long
Dim potential As Double
Dim resistance As Double
Dim pressure As Double
```

```
PulseSeriesGetPipette session, potential, resistance,
pressure
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.33 PulseSeriesGetPotentialType

PulseSeriesGetPotentialType returns the stimulus potential type for the current series.

Syntax

PulseSeriesGetPotentialType *session, type*

session A Long value representing the current session.

type A Long variable to receive the potential type.

Example

```
Dim potential_type As Long

PulseSeriesGetPotentialType session, potential_type
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

The potential type is one of the values listed in table 138.

Table 138 Potential types

Typ	Potential
0	Absolute stimulus
1	Relative stimulus
2	Absolute voltage

9.5.34 PulseSeriesGetRecordingMode

PulseSeriesGetRecordingMode returns the recording mode for the current series.

Syntax

PulseSeriesGetRecordingMode *session, mode*

session A Long value representing the current session.

mode A Long variable to receive the recording mode.

Example

```
Dim mode As Long
```

```
PulseSeriesGetRecordingMode session, mode
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

The recording mode is one of the values listed in table 139.

Table 139 Recording modes

Mode	Protocol
0	In out
1	On cell
2	Out out
3	Whole cell
4	C clamp
5	V clamp

9.5.35 PulseSeriesGetSampleInterval

PulseSeriesGetSampleInterval returns the sampling interval in seconds for the current series.

Syntax

PulseSeriesGetSampleInterval *session, interval*

session A Long value representing the current session.

interval A Double variable to receive the sampling interval.

Example

```
Dim interval As Double
```

```
PulseSeriesGetSampleInterval session, interval
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

All channels are sampled at the same interval. The interval measures the time between samples on the same channel.

9.5.36 PulseSeriesGetSealResistance

PulseSeriesGetSealResistance returns the seal resistance for the current series.

Syntax

PulseSeriesGetSealResistance *session, resistance*

session A Long value representing the current session.

resistance A Double variable to receive the seal resistance.

Example

```
Dim resistance As Double
```

```
PulseSeriesGetSealResistance session, resistance
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.37 PulseSeriesGetSolutions

PulseSeriesGetSolutions returns the solution numbers for the current series.

Syntax

PulseSeriesGetSolutions *session, internal, external*

session A Long value representing the current session.

internal A Long variable to receive the internal solution number.

external A Long variable to receive the external solution number.

Example

```
Dim internal As Long
```

```
Dim external As Long
```

```
PulseSeriesGetSolutions session, internal, external
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.38 PulseSeriesGetStart

PulseSeriesGetStart returns the start time in seconds for the current series.

Syntax

PulseSeriesGetStart *session, start*

session A Long value representing the current session.

start A Double variable to receive the start time.

Example

```
Dim series_start As Double
```

```
PulseSeriesGetStart session, series_start
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

The start time of an series is measured in seconds since the beginning of the experiment.

9.5.39 PulseSeriesGetTemperature

PulseSeriesGetTemperature returns the temperature for the current series.

Syntax

PulseSeriesGetTemperature *session, temperature*

session A Long value representing the current session.

temperature A Double variable to receive the **temperature**.

Example

```
Dim temperature As Double
```

```
PulseSeriesGetTemperature session, temperature
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

9.5.40 PulseSeriesGetTime

PulseSeriesGetTime returns the time of the start of the current series.

Syntax

```
PulseSeriesGetTime session, date_time
```

session A Long value representing the current session.

date_time A Date variable to receive the series start time.

Example

```
Dim date_time As Date
```

```
PulseSeriesGetTime session, date_time
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.41 PulseSeriesGetUserParameter1

PulseSeriesGetUserParameter1 returns the first user parameter for the current series.

Syntax

```
PulseSeriesGetUserParameter1 session, name., value, units
```

session A Long value representing the current session.

name A Variant variable to receive the **parameter name**.

value A Double variable to receive the parameter value.

units A Variant variable to receive the parameter units.

Example

```
Dim name As Variant
```

```
Dim parameter As Double
```

```
Dim units As Variant
```

```
PulseSeriesGetUserParameter1 session, name,  
parameter, units
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.42 PulseSeriesGetUserParameter2

PulseSeriesGetUserParameter2 returns the second user parameter for the current series.

Syntax

PulseSeriesGetUserParameter2 *session, name, value, units*

session A Long value representing the current session.

name A Variant variable to receive the **parameter name**.

value A Double variable to receive the parameter value.

units A Variant variable to receive the parameter units.

Example

```
Dim name As Variant
Dim parameter As Double
Dim units As Variant
```

```
PulseSeriesGetUserParameter2 session, name,
parameter, units
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.43 PulseSeriesSet

PulseSeriesSet selects the current series.

Syntax

PulseSeriesSet *session, series*

session A Long value representing the current session.

series A Long value representing the number of the series to use within the current group.

Example

The following example sets the current series to the last series in the current group.

```
Dim series As Long

PulseSeriesGetCount session, series

If series < 1 Then
    ' Handle the error
End If

PulseSeriesSet session, series
```

Discussion

Series are numbered 1 to N, where N is the number of series in the current group. The number of series in the current group is returned by PulseSeriesGetCount. [☞ PulseSeriesGetCount, p. 306.](#)

9.5.44 PulseSessionClose

PulseSessionClose ends the session. It deallocates any resources allocated DataAccess.

Syntax

```
PulseSessionClose session
```

session A Long value representing the current session.

Example

```
PulseFileClose session  
  
PulseSessionClose session
```

Discussion

PulseSessionClose does not return a value, so it should not be called as a function.

You should close a session to deallocate any resources allocated to the session.

9.5.45 PulseSessionOpen

PulseSessionOpen begins a DataAccess session.

Syntax

```
PulseSessionOpen session
```

session A Long variable to contain the created session.

Example

```
Dim session As Long  
Dim result As Long  
Dim file_name As Variant  
file_name = "Sample.dat"  
  
PulseSessionOpen session  
  
result = PulseFileOpen(session, file_name)
```

Discussion

You must have a session open to use any other DataAccess calls. The handle returned by PulseSessionOpen is a parameter to all other calls.

You can have as many sessions open simultaneously as you would like. For example, if you want to have two data sets open simultaneously, call PulseSessionOpen twice. Use one handle when accessing one data set and the other handle when accessing the other data set.

After you are done with the session, you should call PulseSessionClose. ➤ *PulseSessionClose*, p. 315.

9.5.46 PulseStimulusGetDAC

PulseStimulusGetDAC obtains the physical D/A channel.

Syntax

PulseStimulusGetDAC *session, dac*

session A Long value representing the current session.

dac An Long variable to receive the D/A channel number.

Example

```
Dim dac As Long
```

```
PulseStimulusGetDAC session, dac
```

Discussion

You must have a data set open.

The physical channel number is the physical channel label on the data acquisition device. The channel is one of the values shown in table 140.

Table 140 Stimulus DAC

DAC	Output channel
0	DAC 0
1	DAC 1
2	DAC 2
3	DAC 3
5	Default

If dac is 'Default', then the channel is the default 'V-membrane Out' channel.

9.5.47 PulseStimulusGetRange

PulseStimulusGetRange returns the range of the stimulus.

Syntax

PulseStimulusGetRange *session, minimum, maximum*

session A Long value representing the current session.

minimum A Double variable to receive the lower limit.

maximum A Double variable to receive the upper limit.

Example

```
Dim minimum As Double
```

```
Dim maximum As Double
```

```
PulseStimulusGetRange session, minimum, maximum
```

Discussion

You must have a data set open.

PulseStimulusGetRange returns the minimum and maximum possible values for the stimulus.

9.5.48 PulseStimulusGetTrigger

PulseStimulusGetTrigger returns the parameters for the specified trigger pulse.

Syntax

PulseStimulusGetTrigger *session, trigger, segment, time, duration, value, channel*

session A Long value representing the current session.

trigger A Long value specifying the trigger.

segment A Long variable to receive the trigger segment.

time A Double variable to receive the trigger time.

duration A Double variable to receive the trigger duration.

value A Double variable to receive the trigger value.

channel A Long variable to receive the trigger channel.

Example

```
Dim segment As Long
Dim time As Double
Dim duration As Double
Dim value As Double
Dim channel As Long
```

```
PulseStimulusGetTrigger session, 1, segment, time,
duration, value, channel
```

Discussion

The triggers are numbered 1 to 3. To determine how many triggers are available use PulseStimulusGetTriggerCount. [☞ PulseStimulusGetTriggerCount, p. 318.](#)

Time is the time of the trigger in seconds from the beginning of the segment. Duration is the duration of the trigger in seconds.

Channel is one of the values shown in table 141.

Table 141 Trigger channels

Channel	Output channel
0	DAC 0
1	DAC 1
2	DAC 2
3	DAC 3
4	Digital
5	Default
6	None

If the channel is 'Default', then the channel is the default 'Trigger Out' channel. If the channel is 'None', then there is no trigger output.

For an analog channel, value is the output voltage. For a digital trigger, the lowest byte of value is the digital mask of the channels used. There are eight digital lines.

The first trigger, if enabled, determines when data recording starts. [☞ PulseSegmentGetTrigger, p. 303.](#)

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.49 PulseStimulusGetTriggerCount

PulseStimulusGetTriggerCount returns the number of triggers for the stimulus.

Syntax

PulseStimulusGetTriggerCount *session, count*

session A Long value representing the current session.

count A Long variable to receive the number of triggers for the stimulus.

Example

```
Dim trigger_count As Long
```

```
PulseStimulusGetTriggerCount session, trigger_count
```

Discussion

The trigger count can range between 0 and 3. To get the parameters for a trigger use PulseStimulusGetTrigger. [☞ PulseStimulusGetTrigger, p. 317.](#)

You must have a data set open and the current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

9.5.50 PulseStimulusGetUnits

PulseStimulusGetUnits returns the units for the stimulus.

Syntax

PulseStimulusGetUnits *session, units*

session A Long value representing the current session.

units A Variant variable to receive the **units**.

Example

```
Dim stimulus_units As Variant
```

```
PulseStimulusGetUnits session, stimulus_units
```

Discussion

You must have a data set open.

9.5.51 PulseSweepGetAmplifier

PulseSweepGetAmplifier returns the amplifier settings for the current sweep.

Syntax

```
PulseSweepGetAmplifier session, gain_0, gain_1,
    capacitance, conductance, resistance
```

session A Long value representing the current session.

gain_0 A Double value to receive the gain for the first channel.

gain_1 A Double value to receive the gain for the second channel.

capacitance A Double value to receive the capacitance.

conductance A Double value to receive the series conductance.

resistance A Double value to receive the series resistance compensation.

Example

```
Dim gain_0 As Double
Dim gain_1 As Double
Dim capacitance As Double
Dim conductance As Double
Dim resistance As Double
```

```
PulseSweepGetAmplifier session, gain_0, gain_1,
    capacitance, conductance, resistance
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 326.](#)

The capacitance is the 'C Slow' capacitance in Farads.

The conductance is the 'G Series' conductance in Siemens. The series resistance, 'R Series', is the inverse of the series conductance.

The series resistance compensation is the 'Rs Value' in Ohms.

9.5.52 PulseSweepGetCount

PulseSweepGetCount returns the number of sweeps in the current series.

Syntax

```
PulseSweepGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of sweeps recorded in the current series.

Example

```
Dim sweeps As Long
```

```
PulseSweepGetCount session, sweeps
```

Discussion

You must have a series selected. To set the current series, use PulseSeriesSet. [↗ PulseSeriesSet, p. 314.](#)

Sweeps are numbered 1 to N, where N is the number of sweeps in the series.

9.5.53 PulseSweepGetHoldingPotential

PulseSweepGetHoldingPotential returns the holding potential for the current sweep.

Syntax

PulseSweepGetHoldingPotential *session, potential*

session A Long value representing the current session.

potential A Double variable to the sweep holding potential.

Example

```
Dim potential As Double
```

```
PulseSweepGetHoldingPotential session, potential
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 326.](#)

9.5.54 PulseSweepGetLabel

PulseSweepGetLabel returns the label for the current sweep.

Syntax

PulseSweepGetLabel *session, label*

session A Long value representing the current session.

label A Variant variable to receive the sweep label

Example

```
Dim text As Variant
```

```
PulseSweepGetLabel session, text
```

Discussion

You must have a data set open.

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [↗ PulseSweepSet, p. 326.](#)

9.5.55 PulseSweepGetLeakAvailable

PulseSweepGetLeakAvailable reports whether or not a leak trace is available for the current sweep.

Syntax

PulseSweepGetLeakAvailable *session, available*

session A Long value representing the current session.

available A Long value to receive the leak flag. The value is zero if no leak trace is available for the sweep, and non-zero if a leak trace is available for the sweep.

Example

```
Dim available As Long

PulseSweepGetLeakAvailable session, available

If available <> 0 Then
    ' Process the leak trace
End If
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

9.5.56 PulseSweepGetOnlineResult

PulseSweepGetOnlineResult returns an online result for the current sweep.

Syntax

PulseSweepGetOnlineResult *session, index, x_result, y_result*

session A Long value representing the current session.

index A Long value indicating the result to get.

x_result A Double variable to receive the x result.

y_result A Double variable to receive the y result.

Example

```
Dim x_result As Double
Dim y_result As Double

PulseSweepGetOnlineResult session, 0, x_result,
y_result
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

The index can be 0 or 1.

9.5.57 PulseSweepGetSampleCount

PulseSweepGetSampleCount returns the number of samples in the current sweep.

Syntax

PulseSweepGetSampleCount *session, samples*

session A Long value representing the current session.

samples A Long variable to receive the number of samples recorded in the current sweep.

Example

```
Dim samples As Long
```

```
PulseSweepGetSampleCount session, samples
```

Discussion

The number of samples returned by PulseSweepGetSampleCount is the number of samples recorded for each channel.

The number of samples recorded may be less than the total duration of all stimulus segments if recording does not begin at the beginning of the first stimulus segment. [☞ PulseSegmentGetTrigger, p. 303.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

9.5.58 PulseSweepGetStart

PulseSweepGetStart returns the sweep start time.

Syntax

PulseSweepGetStart *session, sweep_time, data_time*

session A Long value representing the current session.

sweep_time A Double variable to receive the starting time of the sweep.

data_time A Double variable to receive the starting time of the sweep data.

Example

```
Dim sweep_time As Double
```

```
Dim data_time As Double
```

```
PulseSweepGetStart session, sweep_time, data_time
```

Discussion

The sweep time is measured in seconds since the beginning of the series. This is the time the stimulus starts.

The data time is measured in seconds since the beginning of the series. This is the starting time of recorded data. [☞ PulseSegmentGetTrigger, p. 303.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

9.5.59 PulseSweepGetTimer

PulseSweepGetTimer returns the timer value at the start of the sweep.

Syntax

PulseSweepGetTimer *session, timer*

session A Long value representing the current session.

timer A Double variable to receive the timer value at the start of the sweep.

Example

```
Dim timer As Double
```

```
PulseSweepGetTimer session, timer
```

Discussion

The timer value is measured in seconds since the most recent timer reset.

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

9.5.60 PulseSweepGetUserParameter1

PulseSweepGetUserParameter1 returns the first user parameter for the current series.

Syntax

PulseSweepGetUserParameter1 *session, value*

session A Long value representing the current session.

value A Double variable to receive the parameter value.

Example

```
Dim value As Double
```

```
PulseSweepGetUserParameter1 session, value
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

To get the name and units for the parameter, use PulseSeriesGetUserParameter1.

[☞ PulseSeriesGetUserParameter1, p. 313.](#)

9.5.61 PulseSweepGetUserParameter2

PulseSweepGetUserParameter2 returns the first user parameter for the current series.

Syntax

PulseSweepGetUserParameter2 *session, value*

session A Long value representing the current session.

value A Double variable to receive the parameter value.

Example

```
Dim value As Double
```

```
PulseSweepGetUserParameter2 session, value
```

Discussion

The current series must be set. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

To get the name and units for the parameter, use PulseSeriesGetUserParameter2.

[☞ PulseSeriesGetUserParameter2, p. 314.](#)

9.5.62 PulseSweepRead

PulseSweepRead reads the data for a section of one channel of the current sweep into an array.

Syntax

PulseSweepRead *session, channel, start, length, data*

session A Long value representing the current session.

channel A Long value representing the channel containing the data to read.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

data An array of “Double” to receive the data. The array must be long enough to contain all the data in the sweep.

Example

```
Dim samples As Long
Dim buffer() As Double
PulseSweepGetSampleCount session, samples
```

```
If samples > limit Then
    samples = limit
End If
```

```
ReDim buffer(samples)
```

```
PulseSweepRead session, channel, 0, samples,
buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [☞ PulseChannelGetCount, p. 293.](#)

The number of samples in the current sweep is provided by PulseSweepGetSampleCount. [☞ PulseSweepGetSampleCount, p. 322.](#)

9.5.63 PulseSweepReadLeak

PulseSweepReadLeak reads the leak template of the current sweep into an array.

Syntax

PulseSweepReadLeak *session, start, length, data*

session A Long value representing the current session.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

data An array of “Double” to receive the data. The array must be long enough to contain all the data in the leak template.

Example

```
Dim samples As Long
Dim buffer() As Double

PulseSweepGetSampleCount session, samples

ReDim buffer(samples)

PulseSweepReadLeak session, 0, samples, buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

You can use PulseSweepGetLeakAvailable to determine if leak data is available for the current sweep. [☞ PulseSweepGetLeakAvailable, p. 321.](#)

The number of samples in the current sweep is provided by PulseSweepGetSampleCount. [☞ PulseSweepGetSampleCount, p. 322.](#)

9.5.64 PulseSweepReadRaw

PulseSweepRead reads the data without the leak correction for a section of the first channel of the current sweep into an array.

Syntax

PulseSweepReadRaw *session, start, length, data*

session A Long value representing the current session.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

data An array of “Double” to receive the data. The array must be long enough to contain all the data in the sweep.

Example

```
Dim samples As Long
Dim buffer() As Double
PulseSweepGetSampleCount session, samples

If samples > limit Then
    samples = limit
End If

ReDim buffer(samples)

PulseSweepReadRaw session, 0, samples, buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

Channels are numbered 0 to N-1, where N is the number of channels in the current sweep. [☞ PulseChannelGetCount, p. 293.](#)

9.5.65 PulseSweepReadStimulus

PulseSweepReadStimulus reads the output data for the current sweep into an array.

Syntax

PulseSweepReadStimulus *session, start, length, stimulus*

session A Long value representing the current session.

start A Long value representing the first sample of the sweep to read.

length A Long value representing the number of samples to read.

stimulus An array of "Double" to receive the stimulus. The array must be long enough to contain all the data in the sweep stimulus.

Example

```
Dim samples As Long
Dim buffer() As Double
```

```
PulseSweepGetSampleCount session, samples
```

```
ReDim buffer(samples)
```

```
PulseSweepReadStimulus session, 0, samples, buffer(0)
```

Discussion

You must have a sweep selected. To set the current sweep, use PulseSweepSet. [☞ PulseSweepSet, p. 326.](#)

The number of samples in the current sweep is provided by PulseSweepGetSampleCount. [☞ PulseSweepGetSampleCount, p. 322.](#) The returned stimulus samples are those that correspond to the acquired data returned by PulseSweepRead. [☞ PulseSweepRead, p. 324.](#)

9.5.66 PulseSweepSet

PulseSweepSet selects the current sweep.

Syntax

PulseSweepSet *session, sweep*

session A Long value representing the current session.

sweep A Long value representing the sweep to set.

Example

This example sets the current sweep to the last sweep in the current series.

```
Dim sweeps As Long
```

```
PulseSweepGetCount session, sweeps
```

```
If sweeps < 1 Then
    ' Handle the error
End If
```

```
PulseSweepSet session, sweeps
```

Discussion

Sweeps are numbered 1 to N, where N is the number of sweeps in the current series. The number of sweeps in the current series is provided by PulseSweepGetCount. [☞ PulseSweepGetCount, p. 319.](#)

You must have a series selected. To set the current series, use PulseSeriesSet. [☞ PulseSeriesSet, p. 314.](#)

10 SON: Igor Pro

Section	
10.1 Script Interface	P. 327
10.2 Robust Processing	P. 328
10.3 Operations	P. 329
10.4 Reference	P. 330

10.1 Script Interface

This section explains how to access an SON data file using the commands provided by DataAccess. It contains a step by step description of the operations to perform to open a file and read the data in it.

10.1.1 Opening a File

Open a SON data file using the SONFileOpen operation. [☞ SONFileOpen, p. 337.](#)

In order to open a data file, you must supply the full path to the file. The IGOR PRO “Open” command can be used to bring up a file selection dialog and supply the resulting file path.

The following example brings up a file selection dialog to obtain a file specification. It then uses SONFileOpen to open the file for processing:

```
Variable fileReferenceNumber
Variable status

Open /D/R/T="BINFTEXT" fileReferenceNumber

if (strlen(S_fileName) > 0)
    SONFileOpen S_fileName, status
endif
```

10.1.2 File Parameters

Once you have opened a file, you can determine the parameters of the file. The parameters include the creation date and time, the file comment, and the sampling rate.

[☞ File Parameter Operations, p. 330.](#)

The following example obtains a set of file parameters:

```
Variable file_date_time As Date
Variable comment

SONFileGetTime file_date_time, status
SONFileGetComment comment, status
```

10.1.3 Segments

A file is a sequence of segments. You use the segment operations to select an segment and determine its parameters. [☞ Segment Operations, p. 330.](#)

You use SONSegmentGetCount to determine the number of segments in a file. [☞ SONSegmentGetCount, p. 344.](#)

Segments are numbered 0 to N-1, where N is the number of segments in a file.

The following example determines the sample count of each segment in a file.

```
Variable segment_count
Variable start
Variable samples

SONSegmentGetCount channel, segment_count, status

segment = 0
do
```

```

SONSegmentGetRange channel, segment,
  start, samples, status
segment += 1
while (segment < segment_count)

```

10.1.4 Channels

SONChannelGetCount returns the number of channels recorded in a file. [☞ SONChannelGetCount, p. 331.](#)

The following example obtains the number of channels recorded in a file:

```

Variable channel_count

SONChannelGetCount channel_count, status

```

Channel numbers range from 0 to N-1, where N is the number of channels in the file.

10.1.5 Reading Data

To read an segment, use SONSegmentRead. SONSegmentRead can read either a whole segment or sections of an segment. [☞ SONSegmentRead, p. 345.](#)

Normally one would read whole segments but it may be necessary to read sections of an segment if you have insufficient memory available to read whole segments.

Reading Segments

The following example reads each segment into a wave.

```

Variable segment_count
Variable current_segment
Variable segment_start
Variable segment_size

SONSegmentGetCount channel, segment_count, status

current_segment = 0
do
  SONSegmentGetRange
    channel, segment_start, segment_size, status
  Make/R/N=(segment_size) data

  SONSegmentRead channel, 0

```

```

segment_size, data, status
' Data processing goes here
current_segment += 1
while (current_segment < segment_count)

```

Reading Sections of an Segment

The following example reads the first “length” values of the first segment into a wave.

```

Variable segment_start
Variable segment_size

SONSegmentGetRange channel, segment_start,
  segment_size, status

If (segment_size < length)
  ' Limit the read to the actual length of the segment
  length = segment_size
EndIf

Make/R/N=(length) data

SONSegmentRead
  channel, 0, length, data

```

10.1.6 Closing a File

To close a file, use SONFileClose. [☞ SONFileClose, p. 335.](#)

The following example closes an SON file

```

SONFileClose

```

10.2 Robust Processing

if you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

This section explains how to use the information provided by DataAccess to recover from error conditions in your IGOR Pro procedures.

10.2.1 Errors

The status information provided by DataAccess allows you to recover from error conditions caused by the program,

user or the system. It does not allow you to recover from syntax errors in your program.

Program Errors

Errors in your program include such mistakes as using an invalid sweep number. You can determine the range of valid sweep numbers using `SONSegmentGetCount`, so you can avoid using an invalid sweep number. Catching and reporting such errors is helpful in debugging a procedure.

System Errors

System errors include such problems as an error reading an SON data file. Such errors are usually the result of problems such as a corrupted disk, a missing floppy or CD-ROM, or defective hardware such as a disk drive. Catching such errors prevents further problems and may allow the procedure to recover, for example by allowing the user to insert a missing file medium.

User Errors

User errors include mistakes such as when the user selects a file that is not an SON data file. The recovery allows your procedures to provide an error indication to the user and allow the user to select a different file.

Syntax Errors

Syntax errors include mistakes such as passing the wrong number or types of parameters to an operation. An improperly dimensioned wave is also treated as a syntax error. Such errors are returned directly to Igor Pro and cause a procedure to terminate.

10.2.2 Status

Nearly all operations return a status value through a status parameter. The status value is non-zero if an error occurred. All `DataAccess` and system errors are returned in this manner. Such errors will not terminate a procedure. The procedure should be designed to handle such conditions. Igor Pro errors, such as a missing parameter, are returned directly to Igor and will terminate a procedure.

Status values for specific errors are not defined, and may change from version to version of `DataAccess`. The `SONGetStatusText` operation translates a status value to

text. [☞ `SONGetStatusText`, p. 337.](#)

The following example allows the user to select a file, and does not terminate until the user selects a valid SON file.

```

Variable fileNumber
Variable status
String message

do
  Open /D/R/T="BINFTEXT" fileNumber

  if (strlen(S_fileName) > 0)
    SONFileOpen S_fileName, status

    if (status != 0)
      SONGetStatusText status, message
      Print "SONFileOpen error: ", message
    endif
  endif
while (status != 0)

```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

10.3 Operations

This chapter describes each of the operations provided by `DataAccess`. The operations are grouped by category.

10.3.1 Common Operations

The common operations are valid regardless of whether a file is open or not. The common operations are shown in table 142.

Table 142 Common operations

Operation	
<code>SONGetVersion</code>	p. 338
<code>SONGetStatusText</code>	p. 337

10.3.2 File Operations

The file operations allow you to access an SON data file.

The file operations are shown in table 143.

Table 143 File operations

Operation	
SONFileOpen	p. 337
SONFileClose	p. 335

All remaining operations require a file to be open.

10.3.3 File Parameter Operations

The file parameter operations allow you to obtain file parameters. The file parameter operations are shown in table 144.

Table 144 File parameter operations

Operation	
SONFileGetComment	p. 335
SONFileGetTime	p. 336
SONFileGetVersion	p. 336

10.3.4 Segment Operations

The segment operations allow you to access each segment of a file. The segment operations are shown in table 145.

Table 145 Segment operations

Operation	
SONSegmentGetCount	p. 344
SONSegmentGetRange	p. 344
SONSegmentRead	p. 345

10.3.5 Channel Operations

The channel operations allow you to determine the number of channels in a file, and obtain channel parameters. The channel operations are shown in table 146.

Table 146 Channel operations

Operation	
SONChannelGetCount	p. 331
SONChannelGetLabel	p. 332
SONChannelGetMaximumTime	p. 332
SONChannelGetPreTrigger	p. 333
SONChannelGetSampleInterval	p. 333

Table 146 Channel operations

Operation	
SONChannelGetType	p. 334
SONChannelGetUnits	p. 334

10.3.6 Marker Operations

The marker operations allow you to read the marker stored in a data file. The marker operations are shown in table 147.

Table 147 Marker operations

Operation	
SONMarkerGetCount	p. 338
SONMarkerGetCurve	p. 339
SONMarkerGetCurveCount	p. 340
SONMarkerGetCurveSize	p. 340
SONMarkerGetData	p. 341
SONMarkerGetNext	p. 341
SONMarkerGetText	p. 342
SONMarkerGetTimes	p. 342
SONMarkerGetValueCount	p. 343
SONMarkerGetValues	p. 343

10.4 Reference

This section lists each DataAccess operation in alphabetical order.

10.4.1 SONChannelGetComment

SONChannelGetComment returns the comment for the specified channel.

Syntax

SONChannelGetComment *channel, comment, status*

comment A String variable to receive the channel comment.

status A String variable to receive the status value.

Example

String comment

SONChannelGetComment 0, comment, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

10.4.2 SONChannelGetCount

SONChannelGetCount returns the number of channels recorded in the current SON data file.

Syntax

SONChannelGetCount *count, status.*

count A variable to receive the number of channels.

status A variable to receive the status value.

Example

Variable channel_count

SONChannelGetCount channel_count, status

Discussion

You must have a file open.

10.4.3 SONChannelGetLabel

SONChannelGetLabel returns the label for the specified channel.

Syntax

```
SONChannelGetLabel channel, label, status
```

label A String variable to receive the channel label.

status A String variable to receive the status value.

Example

```
String label
```

```
SONChannelGetLabel label, status
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

10.4.4 SONChannelGetMaximumTime

SONChannelGetMaximumTime returns maximum time data is recorded for the specified channel.

Syntax

```
SONChannelGetMaximumTime channel, time, status
```

channel A value representing the channel to use.

time A variable to receive the maximum time.

status A variable to receive the status value.

Example

```
Variable id
```

```
SONChannelGetMaximumTime 1, id, status
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The maximum time is in seconds from the file creation time.

10.4.5 SONChannelGetPreTrigger

SONChannelGetPreTrigger the number of pre-trigger samples recorded for the markers.

Syntax

SONChannelGetPreTrigger *channel, samples, status*

channel A value specifying the channel.

samples A variable to receive the number of pre-trigger samples.

status A variable to receive the status value.

Example

Variable channel_count

SONChannelGetPreTrigger channel_count, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 334.](#)

10.4.6 SONChannelGetSampleInterval

SONChannelGetSampleInterval returns the sampling interval in seconds.

Syntax

SONChannelGetSampleInterval *channel, interval, status*

interval A variable to receive the sampling interval.

status A variable to receive the status value.

Example

Variable interval

SONChannelGetSampleInterval interval, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The interval is in seconds.

10.4.7 SONChannelGetType

SONChannelGetType returns the type of the specified channel.

Syntax

SONChannelGetType *channel, type, status*

channel A value specifying the channel.

type A variable to receive the channel type.

status A variable to receive the status value.

Example

Variable channel_type

SONChannelGetType 0, channel_type, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The type can be one of the following.

Table 148 Channel types

Typ e	Description
1	Adc
2	EventFall
3	EventRise
4	EventBoth
5	Marker
6	AdcMark
7	RealMark
8	TextMark
9	RealMark

10.4.8 SONChannelGetUnits

SONChannelGetUnits returns the units of the specified channel.

Syntax

SONChannelGetUnits *channel, units, status*

channel A value specifying the channel.

units A String variable in which the channel units are returned.

status A String variable to receive the status value.

Example

String text

SONChannelGetUnits 0, text, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

10.4.9 SONFileClose

SONFileClose terminates processing of an SON file.

Syntax

```
SONFileClose
```

Example

```
SONFileClose
```

Discussion

You must have a file open.

You should perform SONFileClose as soon as you complete processing of a file.

10.4.10 SONFileGetComment

SONFileGetComment returns the specified comment for the file.

Syntax

```
SONFileGetComment index, comment, status
```

index A value specifying the index.

comment A string in which the file comment is returned.

status A variable to receive the status value.

Example

```
String text
```

```
SONGetFileComment 0, text, status
```

Discussion

You must have a file open.

The index can range between 0 and 4.

10.4.11 SONFileGetTime

SONFileGetTime returns the creation date and time of the file.

Syntax

SONFileGetTime *date_time, status*

date_time A Date variable to receive the file creation date and time.

status A variable to receive the status value.

Example

Variable date_time As Date

SONFileGetTime date_time, status

Discussion

You must have a file open.

10.4.12 SONFileGetVersion

SONFileGetVersion returns the Spike2 SON format version of the file.

Syntax

SONFileGetVersion *version, status*

version A String variable in which the file version is returned.

status A variable to receive the status value.

Example

Variable status

String version

SONFileGetVersion version, status

Discussion

You must have a file open.

10.4.13 SONFileOpen

SONFileOpen prepares an SON data file for processing.

Syntax

SONFileOpen *path, status*

path A String value representing the file path.

status A variable to receive the status value.

Example

String file

```
file = "Data.dat"
SONFileOpen file, status
```

Discussion

You must not have a file open.

SONFileOpen loads the SON file header into memory. This header describes all the parameters of the file.

Once a file is open, you can access file parameters. You can process the segments in the file. [☞ Segment Operations, p. 330.](#) [☞ File Parameter Operations, p. 330.](#)

When you are done with a file, you should close it. [☞ SONFileClose, p. 335.](#)

10.4.14 SONGetStatusText

SONGetStatusText translates a status value returned by a DataAccess operation to a text string.

Syntax

SONGetStatusText *status, message*

status A value representing the status value to translate.

message A String variable in which the translated text is returned.

Example

```
Variable status
String message
Variable file_name
file_name = "Sample.dat"
```

```
SONFileOpen file_name, status
```

```
If status <> 0 Then
  SONGetStatusText status, message
  ' Display the message here
End If
```

Discussion

SONGetStatusText does not return a value, so it should not be called as a function.

You need not have a file open.

SONGetStatusText is the only means provided to interpret status values.

10.4.15 SONGetVersion

SONGetVersion returns the version number of the DataAccess package.

Syntax

```
SONGetVersion version
```

version A variable to receive the DataAccess version number.

Example

```
Variable version
```

```
SONGetVersion version
```

Discussion

SONGetVersion does not return a value, so it should not be called as a function.

You need not have a file open.

If you are writing a set of procedures using DataAccess, you can use SONGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess. You can use the following code to ensure that this version or a later version is in use:

```
Variable version
SONGetVersion version
```

```
If version < N Then
    ' Handle the error
End If
```

10.4.16 SONMarkerGetCount

SONMarkerGetCount returns the number of markers recorded in the current SON data file.

Syntax

```
SONMarkerGetCount channel, count, status
```

channel A value specifying the channel.

count A variable to receive the number of markers.

status A variable to receive the status value.

Example

```
Variable channel_count
```

```
SONMarkerGetCount channel_count, status
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

10.4.17 SONMarkerGetCurve

SONMarkerGetCurve reads the curve from the specified marker for an AdcMark channel.

Syntax

SONMarkerGetCurve *channel, marker, curve, data, status*

channel A value specifying the channel.

marker A value specifying the marker.

curve A value specifying the curve.

data The name of a wave to receive the data.

status A variable to receive the status value.

Example

```
Variable data()
Variable curve_size
```

```
SONMarkerGetCurveSize 0, curve_size
Make/D/N=(curve_size) data
```

```
SONMarkerGetCurve 0, 0, 0, data, status
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 334.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 338.](#)

Curves are numbered 0 to N-1, where N is the number of curves recorded for the channel. To determine the number of curves for the channel, use SONMarkerGetCurveCount. [☞ SONMarkerGetCurveCount, p. 340.](#)

The wave is assumed to be large enough to hold the entire curve. Use SONMarkerGetCurveSize to determine the curve size. [☞ SONMarkerGetCurveSize, p. 340.](#)

The wave must consist of four byte real values.

10.4.18 SONMarkerGetCurveCount

SONMarkerGetCurveCount returns the number of curves recorded for the specified channel.

Syntax

SONMarkerGetCurveCount *channel, count, status*

channel A value specifying the channel.

count A variable to receive the number of curves.

status A variable to receive the status value.

Example

Variable curve_count

SONMarkerGetCurveCount 0, curve_count, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 334.](#)

10.4.19 SONMarkerGetCurveSize

SONMarkerGetCurveSize returns the number of samples recorded in each marker curve.

Syntax

SONMarkerGetCurveSize *channel, size, status*

channel A value specifying the channel.

size A variable to receive the curve size.

status A variable to receive the status value.

Example

Variable size

SONMarkerGetCurveSize 0, size, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 334.](#)

10.4.20 SONMarkerGetData

SONMarkerGetData reads the marker data.

Syntax

```
SONMarkerGetData channel, first_marker, data,
count, status
```

channel A value specifying the channel.

first_marker A value specifying the first marker.

data A wave to receive the data.

count A value specifying the number of markers to read.

status A variable to receive the status value.

Example

Variable count

```
SONMarkerGetCount 0, count
Make/D/N=(count) data
```

```
SONMarkerGetData 0, 0, data, count, status
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 338.](#)

The channel must be an Marker, AdcMark, RealMark or TextMark channel. [☞ SONChannelGetType, p. 334.](#)

The wave must consist of four byte integer values.

10.4.21 SONMarkerGetNext

SONMarkerGetNext returns the index of the first marker at or following the specified time.

Syntax

```
SONMarkerGetNext channel, time, marker, status
```

channel A value specifying the channel.

time A value specifying a time.

marker A variable to receive the next marker.

status A variable to receive the status value.

Example

Variable marker

```
SONMarkerGetNext 0, 10.0, marker, status
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 338.](#)

The time is seconds from the file creation time.

10.4.22 SONMarkerGetText

SONMarkerGetText returns the text for the specified marker.

Syntax

SONMarkerGetText *channel, marker, text, status*

channel A value specifying the channel.

marker A value specifying the marker.

text A String variable to receive the text.

status A variable to receive the status value.

Example

String text

SONMarkerGetText 0, 0, text, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 338.](#)

The channel must be an TextMark channel. [☞ SONChannelGetType, p. 334.](#)

10.4.23 SONMarkerGetTimes

SONMarkerGetTimes reads the marker times for the specified channel.

Syntax

SONMarkerGetTimes *channel, first_marker, times, count, status*

channel A value specifying the channel.

first_marker A value specifying the first marker.

times A waveto receive the times.

count A value specifying the number of markers to read.

status A variable to receive the status value.

Example

Variable count

SONMarkerGetCount 0, count, status
Make/D/N=(count) times

SONMarkerGetTimes 0, 0, times, count, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 338.](#)

The wave must consist of eight byte double real values.

The times are in seconds from the file creation time.

10.4.24 SONMarkerGetValueCount

SONMarkerGetValueCount returns the size of the value array stored with the markers.

Syntax

SONMarkerGetValueCount *channel, count, status*

channel A value specifying the channel.

count A variable to receive the marker value count.

status A variable to receive the status value.

Example

Variable value_count

SONMarkerGetValueCount 0, value_count, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an RealMark channel. [☞ SONChannelGetType, p. 334.](#)

10.4.25 SONMarkerGetValues

SONMarkerGetValues reads the values for the specified RealMark channel.

Syntax

SONMarkerGetValues *channel, first_marker, values, count, status*

channel A value specifying the channel.

first_marker A value specifying the first marker.

values A wave to receive the values.

count A value specifying the number of markers to read.

status A variable to receive the status value.

Example

Variable count

SONMarkerGetCount 0, count, status
Make/D/N=(count) values

SONMarkerGetValues 0, 0, values, count, status

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 338.](#)

The channel must be an RealMark channel with a value array of size one. [☞ SONChannelGetType, p. 334.](#)
[☞ SONMarkerGetValueCount, p. 343.](#)

The wave must consist of four byte real values.

10.4.26 SONSegmentGetCount

SONSegmentGetCount returns the number of segments recorded for the channel.

Syntax

SONSegmentGetCount *channel, count, status*

channel A value specifying the channel.

count A variable in which to store the number of segments.

status A variable to receive the status value.

Example

Variable `segment_count`

SONSegmentGetCount `segment_count, status`

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an Adc or RealWave channel. [☞ SONChannelGetType, p. 334.](#)

The segment count is the number of segments recorded in the SON data file. Segments are numbered 0 to N-1, where N is the number of segments in the file.

10.4.27 SONSegmentGetRange

SONSegmentGetRange returns the range of the specified segment.

Syntax

SONSegmentGetRange *channel, segment, start, sample_count, status*

channel A value specifying the channel.

start A variable in which to return the segment start time.

segment A value specifying the segment.

sample_count A variable in which to return the segment sample count.

status A variable to receive the status value.

Example

Variable `start`

Variable `sample_count`

SONSegmentGetRange `0, 0, start, sample_count, status`

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an Adc or RealWave channel. [☞ SONChannelGetType, p. 334.](#)

Segments are numbered 0 to N-1, where N is the number of segments recorded in the file. To determine the number of segments in the file, use SONSegmentGetCount. [☞ SONSegmentGetCount, p. 344.](#)

The start of a sweep is measured in seconds from the file creation time.

10.4.28 SONSegmentRead

SONSegmentRead reads the data for the specified channel and segment.

Syntax

```
SONSegmentRead
    channel, segment, start, length, data, status
```

channel A value specifying the channel.

segment A value specifying the segment.

start A value representing the first sample to read.

length A value representing the number of samples to read.

data A wave to receive the data. The wave must be long enough to contain all the data in the section.

status A variable to receive the status value.

Example

```
Variable samples
```

```
Variable start
```

```
SONSegmentGetSampleCount 0, 0, samples, status
```

```
samples = samples/2
```

```
start = samples/2
```

```
Make/D/N=(samples) data
```

```
SONSegmentRead 0, 0, start, samples, data, status
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 331.](#)

The channel must be an Adc or RealWave channel. [☞ SONChannelGetType, p. 334.](#)

Segments are numbered 0 to N-1, where N is the number

of segments recorded in the file. To determine the number of segments in the file, use SONSegmentGetCount. [☞ SONSegmentGetCount, p. 344.](#)

The wave must consist of four byte real values.

The start and length values must satisfy the following criteria:

- 1 The start value must not be negative.
- 2 The length value must be greater than zero.

11 SON: Visual Basic

Section	
11.1 Using DataAccess	P. 346
11.2 Getting Started	P. 346
11.3 Robust Processing	P. 348
11.4 Operations	P. 348
11.5 Reference	P. 350

11.1 Using DataAccess

To make DataAccess available to a Visual Basic program, add the file SONVB.bas to the project. Ensure that the file SONVB.dll is in your path.

To make DataAccess available to an Systat Software SigmaPlot script module, include the following statement in the module:

```
'#Uses "SONVB.bas"
```

The leading quote (!) is required. The bas file must be in the same folder as your SigmaPlot project, or you must explicitly specify the path to the bas file.

11.1.1 Status

Each operation returns a status code. This code is zero if no error is detected.

If an error is detected, the returned status value is non-zero. It can be translated to a text string using SONGetStatusText.

11.1.2 Data Types

Integer values are passed as the data type "Long". Real values are passed as the data type "Double". Arrays are passed by passing the first element in the array. Character string values are passed as the data type "Variant". The data type "String" is not used because when Visual Basic passes parameters to dlls, it performs undesired translations from Unicode to ASCII.

11.1.3 Calls

Almost all DataAccess procedures return a value. In Visual Basic, you have several choices of syntax when you call such procedures. For example, you can call SONChannelGetCount as follows:

- 1 As a function, using the return value:

```
result = SONChannelGetCount(session, count)
```

- 2 As a subroutine using "Call":

```
Call SONChannelGetCount(session, count)
```

- 3 Directly as a command. In this case, the parameters are not enclosed in parentheses:

```
SONChannelGetCount session, count
```

Any of the methods work. The examples in this manual generally use the command form.

11.2 Getting Started

This section explains how to access an SON data file. It contains a step by step description of the operations to perform to open a file and read the data in it.

11.2.1 Opening a Session

Open a session using the SONSessionOpen operation.  *SONSessionOpen*, p. 364.

SONSessionOpen operation returns a handle. You must pass this handle to other operations that use that session.

11.2.2 Opening a File

Open an SON data file using the SONFileOpen operation. [☞ SONFileOpen, p. 357.](#)

In order to open a data file, you must supply a file path.

The following example opens a session, then opens a file with the path “f:\test.dat”. After the file is open, it closes the file and the session.

```
Dim session As Long
Dim status As Long
Dim file_name As Variant
file_name = "f:\test.dat"

SONSessionOpen session
status = SONFileOpen(session, file_name)

If status <> 0 Then
    ' The file could not be opened. Handle the
    ' error and do not call SONFileClose
End If

SONFileClose session
SONSessionClose session
```

11.2.3 File Parameters

Once you have opened a file, you can determine the parameters of the file. The parameters include the creation date and time, the file comment, and the sampling rate. [☞ File Parameter Operations, p. 349.](#)

The following example obtains a set of file parameters:

```
Dim file_date_time As Date
Dim comment As Variant

SONFileGetTime session, file_date_time
SONFileGetComment session, comment
```

11.2.4 Segments

A file is a sequence of segments. You use the segment operations to select an segment and determine its parameters.

[☞ Segment Operations, p. 349.](#)

You use SONSegmentGetCount to determine the number of segments in a file. [☞ SONSegmentGetCount, p. 365.](#)

Segments are numbered 0 to N-1, where N is the number of segments in a file.

The following example determines the sample count of each segment in a file.

```
Dim segment_count As Long
Dim start As Double
Dim samples As Long

SONSegmentGetCount session, channel, segment_count

For segment = 0 To segment_count - 1
    SONSegmentGetRange session, channel, segment,
        start, samples
Next sweep
```

11.2.5 Channels

SONChannelGetCount returns the number of channels recorded in a file. [☞ SONChannelGetCount, p. 351.](#)

The following example obtains the number of channels recorded in a file:

```
Dim channel_count As Long

SONChannelGetCount session, channel_count
```

Channel numbers range from 0 to N-1, where N is the number of channels in the file.

11.2.6 Reading Data

To read an segment, use SONSegmentRead. SONSegmentRead can read either a whole segment or sections of an segment. [☞ SONSegmentRead, p. 366.](#)

Normally one would read whole segments but it may be necessary to read sections of an segment if you have insufficient memory available to read whole segments.

Reading Segments

The following example reads each segment into an array.

```
Dim buffer() As Double
Dim segment_count As Long
Dim current_segment As Long
Dim segment_start As Long
Dim segment_size As Long

SONSegmentGetCount session, channel, segment_count

For current_segment = 0 To segment_count - 1
    SONSegmentGetRange
        session, channel, segment_start, segment_size
    ReDim buffer(segment_size-1)

    SONSegmentRead session, channel, 0,
        segment_size, buffer(0)
    ' Data processing goes here
Next current_segment
```

Reading Sections of an Segment

The following example reads the first “length” values of the first segment into an array.

```
Dim buffer() As Double
Dim segment_start As Long
Dim segment_size As Long

SONSegmentGetRange session, channel, segment_start,
    segment_size

If segment_size < length Then
    ' Limit the read to the actual length of the segment
    length = segment_size
End If

ReDim buffer(length-1)

SONSegmentRead
    session, channel, 0, length, buffer
```

11.2.7 Closing a File

To close a file, use SONFileClose. *☞ SONFileClose, p. 354.*

The following example closes an SON file, then closes the

associated session.

```
SONFileClose session
SONSessionClose session
```

11.3 Robust Processing

If you write programs to be used by others, you may find it helpful to handle error conditions, recovering in a manner that allows the user to proceed with their operations.

All operations return a status value of type Long. The value is zero if the operation was successful, and non-zero if an error was detected.

If an error is detected, you can use SONGetStatusText to translate the status code to a message. *☞ SONGetStatusText, p. 363.*

The following example shows how the user might be allowed to select a file. The example does not terminate until the user selects a valid SON file.

```
Dim filespec As Variant
Dim status As Long
Dim session As Long

SONSessionOpen session

Do
    ' Add code here to obtain the file specification in "filespec"
    status = SONFileOpen(session, filespec)

    If status = 0 Then
        Exit Do
    End If

    ' Translate the status code to a message here and
    ' display it for the user
Loop
```

This example is not complete, because you should provide some way for the user to exit the loop without opening a file at all.

11.4 Operations

This chapter describes each of the operations provided by

DataAccess. The operations are grouped by category.

11.4.1 Common Operations

The common operations are valid regardless of whether a session is open or not. The common operations are shown in table 149.

Table 149 Common operations

Operation	
SONGetVersion	p. 363
SONGetStatusText	p. 363

11.4.2 Session Operations

The session operations allow you to open and close a session. The session operations are shown in table 150.

Table 150 Session Operations

Operation	
SONSessionOpen	p. 364
SONSessionClose	p. 364

All other operations require a session to be open.

11.4.3 File Operations

The file operations allow you to access an SON data file. The file operations are shown in table 151.

Table 151 File operations

Operation	
SONFileOpen	p. 357
SONFileClose	p. 354

All remaining operations require a file to be open.

11.4.4 File Parameter Operations

The file parameter operations allow you to obtain file parameters. The file parameter operations are shown in

table 152.

Table 152 File parameter operations

Operation	
SONFileGetComment	p. 355
SONFileGetSignature	p. 355
SONFileGetTime	p. 356
SONFileGetVersion	p. 356

11.4.5 Segment Operations

The segment operations allow you to access each segment of a file. The segment operations are shown in table 153.

Table 153 Segment operations

Operation	
SONSegmentGetCount	p. 365
SONSegmentGetRange	p. 365
SONSegmentRead	p. 366

11.4.6 Channel Operations

The channel operations allow you to determine the number of channels in a file, and obtain channel parameters. The channel operations are shown in table 154.

Table 154 Channel operations

Operation	
SONChannelGetCount	p. 351
SONChannelGetLabel	p. 351
SONChannelGetMaximumTime	p. 352
SONChannelGetPreTrigger	p. 352
SONChannelGetSampleInterval	p. 353
SONChannelGetType	p. 353
SONChannelGetUnits	p. 354

11.4.7 Marker Operations

The marker operations allow you to read the marker stored in a data file. The marker operations are shown in

table 155.

Table 155 Marker operations

Operation	
SONMarkerGetCount	p. 357
SONMarkerGetCurve	p. 358
SONMarkerGetCurveCount	p. 359
SONMarkerGetCurveSize	p. 359
SONMarkerGetData	p. 360
SONMarkerGetNext	p. 360
SONMarkerGetText	p. 361
SONMarkerGetTimes	p. 361
SONMarkerGetValueCount	p. 362
SONMarkerGetValues	p. 362

11.5 Reference

This section lists each DataAccess operation in alphabetical order.

11.5.1 SONChannelGetComment

SONChannelGetComment returns the comment for the specified channel.

Syntax

SONChannelGetComment *session, channel, comment*

session A Long value representing the current session.

comment A Variant variable to receive the channel comment.

Example

```
Dim comment As Variant
```

```
SONChannelGetComment session, 0, comment
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount.  *SONChannelGetCount*, p. 351.

11.5.2 SONChannelGetCount

SONChannelGetCount returns the number of channels recorded in the current SON data file.

Syntax

```
SONChannelGetCount session, count
```

session A Long value representing the current session.

count A Long variable to receive the number of channels.

Example

```
Dim channel_count As Long  
  
SONChannelGetCount session, channel_count
```

Discussion

You must have a file open.

11.5.3 SONChannelGetLabel

SONChannelGetLabel returns the label for the specified channel.

Syntax

```
SONChannelGetLabel session, channel, label
```

session A Long value representing the current session.

label Variant variable to receive the channel label.

Example

```
Dim label As Variant  
  
SONChannelGetLabel session, label
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount.  *SONChannelGetCount, p. 351.*

11.5.4 SONChannelGetMaximumTime

SONChannelGetMaximumTime returns maximum time data is recorded for the specified channel.

Syntax

SONChannelGetMaximumTime *session, channel, time*

session A Long value representing the current session.

channel A Long value representing the channel to use.

time A Double variable to receive the maximum time.

Example

```
Dim id As Long
```

```
SONChannelGetMaximumTime session, 1, id
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The maximum time is in seconds from the file creation time.

11.5.5 SONChannelGetPreTrigger

SONChannelGetPreTrigger the number of pre-trigger samples recorded for the markers.

Syntax

SONChannelGetPreTrigger *session, channel, samples*

session A Long value representing the current session.

channel A Long value specifying the channel.

samples A Long variable to receive the number of pre-trigger samples.

Example

```
Dim channel_count As Long
```

```
SONChannelGetPreTrigger session, channel_count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 353.](#)

11.5.6 SONChannelGetSampleInterval

SONChannelGetSampleInterval returns the sampling interval in seconds.

Syntax

SONChannelGetSampleInterval *session, channel, interval*

session A Long value representing the current session.

interval A Double variable to receive the sampling interval.

Example

```
Dim interval As Double
```

```
SONChannelGetSampleInterval session, interval
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The interval is in seconds.

11.5.7 SONChannelGetType

SONChannelGetType returns the type of the specified channel.

Syntax

SONChannelGetType *session, channel, type*

session A Long value representing the current session.

channel A Long value specifying the channel.

type A Long variable to receive the channel type.

Example

```
Dim channel_type As Long
```

```
SONChannelGetType session, 0, channel_type
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The type can be one of the following.

Table 156 Channel types

Type	Description
1	Adc
2	EventFall
3	EventRise
4	EventBoth
5	Marker
6	AdcMark
7	RealMark
8	TextMark
9	RealWave

11.5.8 SONChannelGetUnits

SONChannelGetUnits returns the units of the specified channel.

Syntax

SONChannelGetUnits *session, channel, units*

session A Long value representing the current session.

channel A Long value specifying the channel.

units A Variant variable in which the channel units are returned.

Example

Dim text As Variant

SONChannelGetUnits session, 0, text

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

11.5.9 SONFileClose

SONFileClose terminates processing of an SON file.

Syntax

SONFileClose *session*

session A Long value representing the current session.

Example

SONFileClose session
SONSessionClose session

Discussion

You must have a file open.

You should perform SONFileClose as soon as you complete processing of a file.

11.5.10 SONFileGetComment

SONFileGetComment returns the specified comment for the file.

Syntax

SONFileGetComment *session, index, comment*

session A Long value representing the current session.

index A Long value specifying the index.

comment A Variant variable in which the file comment is returned.

Example

```
Dim text As Variant
```

```
SONGetFileComment session, 0, text
```

Discussion

You must have a file open.

The index can range between 0 and 4.

11.5.11 SONFileGetSignature

SONFileGetSignature returns a signature that uniquely identifies the file.

Syntax

SONFileGetSignature *session, signature*

session A Long value representing the current session.

signature A Variant variable in which the file signature is returned.

Example

```
Dim signature As Variant
```

```
SONFileGetSignature session, signature
```

Discussion

You must have a file open.

The returned signature is a string containing a 32 character hexadecimal value.

The signature is calculated from information in the file. Except for a very unlikely coincidence, the signature should uniquely identify the file.

11.5.12 SONFileGetTime

SONFileGetTime returns the creation date and time of the file.

Syntax

```
SONFileGetTime session, date_time
```

session A Long value representing the current session.

date_time A Date variable to receive the file creation date and time.

Example

```
Dim date_time As Date
```

```
SONFileGetTime session, date_time
```

Discussion

You must have a file open.

11.5.13 SONFileGetVersion

SONFileGetVersion returns the Spike2 SON format version of the file.

Syntax

```
SONFileGetVersion session, version
```

session A Long value representing the current session.

version A Variant variable in which the file version is returned.

Example

```
Dim version As Variant
```

```
SONFileGetVersion session, version
```

Discussion

You must have a file open.

11.5.14 SONFileOpen

SONFileOpen prepares an SON data file for processing.

Syntax

```
SONFileOpen session, path
```

session A Long value representing the current session.

path A Variant value representing the file path.

Example

```
Dim file As Variant
file = "Data.dat"
SONFileOpen session, file
```

Discussion

You must not have a file open.

SONFileOpen loads the SON file header into memory. This header describes all the parameters of the file.

Once a file is open, you can access file parameters. You can process the segments in the file. [☞ Segment Operations, p. 349.](#) [☞ File Parameter Operations, p. 349.](#)

When you are done with a file, you should close it. [☞ SONFileClose, p. 354.](#)

11.5.15 SONMarkerGetCount

SONMarkerGetCount returns the number of markers recorded in the current SON data file.

Syntax

```
SONMarkerGetCount session, channel, count
```

session A Long value representing the current session.

channel A Long value specifying the channel.

count A Long variable to receive the number of markers.

Example

```
Dim channel_count As Long
SONMarkerGetCount session, channel_count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

11.5.16 SONMarkerGetCurve

SONMarkerGetCurve reads the curve from the specified marker for an AdcMark channel.

Syntax

SONMarkerGetCurve *session, channel, marker, curve, data*

session A Long value representing the current session.

channel A Long value specifying the channel.

marker A Long value specifying the marker.

curve A Long value specifying the curve.

data The first element of a Real array to receive the curve data.

Example

```
Dim data() As Real
Dim curve_size As Long

SONMarkerGetCurveSize session, 0, curve_size
ReDim data(curve_size)

SONMarkerGetCurve
  session, 0, 0, data(0)
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 353.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 357.](#)

Curves are numbered 0 to N-1, where N is the number of curves recorded for the channel. To determine the number of curves for the channel, use SONMarkerGetCurveCount.

[☞ SONMarkerGetCurveCount, p. 359.](#)

The buffer is assumed to be large enough to hold the entire curve. Use SONMarkerGetCurveSize to determine the curve size. [☞ SONMarkerGetCurveSize, p. 359.](#)

11.5.17 SONMarkerGetCurveCount

SONMarkerGetCurveCount returns the number of curves recorded for the specified channel.

Syntax

SONMarkerGetCurveCount *session, channel, count*

session A Long value representing the current session.

channel A Long value specifying the channel.

count A Long variable to receive the number of curves.

Example

```
Dim curve_count As Long
```

```
SONMarkerGetCurveCount session, 0, curve_count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 353.](#)

11.5.18 SONMarkerGetCurveSize

SONMarkerGetCurveSize returns the number of samples recorded in each marker curve.

Syntax

SONMarkerGetCurveSize *session, channel, size*

session A Long value representing the current session.

channel A Long value specifying the channel.

size A Long variable to receive the curve size.

Example

```
Dim size As Long
```

```
SONMarkerGetCurveSize session, 0, size
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The channel must be an AdcMark channel. [☞ SONChannelGetType, p. 353.](#)

11.5.19 SONMarkerGetData

SONMarkerGetData reads the marker data.

Syntax

```
SONMarkerGetData session, channel, first_marker, data,
count
```

session A Long value representing the current session.

channel A Long value specifying the channel.

first_marker A Long value specifying the first marker.

data The first element of a Long array to receive the data.

count A Long value specifying the number of markers to read.

Example

```
Dim data() As Long
Dim count As Long
```

```
SONMarkerGetCount session, 0, count
ReDim data(count)
```

```
SONMarkerGetData
session, 0, 0, data(0), count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 357.](#)

The channel must be an Marker, AdcMark, RealMark or TextMark channel. [☞ SONChannelGetType, p. 353.](#)

11.5.20 SONMarkerGetNext

SONMarkerGetNext returns the index of the first marker at or following the specified time.

Syntax

```
SONMarkerGetNext session, channel, time, marker
```

session A Long value representing the current session.

channel A Long value specifying the channel.

time A Double value specifying a time.

marker A Long variable to receive the next marker.

Example

```
Dim marker As Long
```

```
SONMarkerGetNext session, 0, 10.0, marker
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 357.](#)

The time is seconds from the file creation time.

11.5.21 SONMarkerGetText

SONMarkerGetText returns the text for the specified marker.

Syntax

SONMarkerGetText *session, channel, marker, text*

session A Long value representing the current session.

channel A Long value specifying the channel.

marker A Long value specifying the marker.

text A Variant variable to receive the text.

Example

```
Dim text As Variant
```

```
SONMarkerGetText session, channel_count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 357.](#)

The channel must be an TextMark channel. [☞ SONChannelGetType, p. 353.](#)

11.5.22 SONMarkerGetTimes

SONMarkerGetTimes reads the marker times for the specified channel.

Syntax

SONMarkerGetTimes *session, channel, first_marker, times, count*

session A Long value representing the current session.

channel A Long value specifying the channel.

first_marker A Long value specifying the first marker.

times The first element of a Double array to receive the times.

count A Long value specifying the number of markers to read.

Example

```
Dim times() As Double
```

```
Dim count As Long
```

```
SONMarkerGetCount session, 0, count
```

```
ReDim data(count)
```

```
SONMarkerGetTimes
```

```
session, 0, 0, times(0), count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 357.](#)

The times are in seconds from the file creation time.

11.5.23 SONMarkerGetValueCount

SONMarkerGetValueCount returns the size of the value array stored with the markers.

Syntax

SONMarkerGetValueCount *session, channel, count*

session A Long value representing the current session.

channel A Long value specifying the channel.

count A Long variable to receive the marker value count.

Example

```
Dim value_count As Long
```

```
SONMarkerGetValueCount session, 0, value_count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The channel must be an RealMark channel. [☞ SONChannelGetType, p. 353.](#)

11.5.24 SONMarkerGetValues

SONMarkerGetValues reads the values for the specified RealMark channel.

Syntax

SONMarkerGetValues *session, channel, first_marker, values, count*

session A Long value representing the current session.

channel A Long value specifying the channel.

first_marker A Long value specifying the first marker.

values The first element of a Real array to receive the values.

count A Long value specifying the number of markers to read.

Example

```
Dim values() As Real
```

```
Dim count As Long
```

```
SONMarkerGetCount session, 0, count
```

```
ReDim data(count)
```

```
SONMarkerGetValues
```

```
session, 0, 0, values(0), count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

Markers are numbered 0 to N-1, where N is the number of markers recorded in the file. To determine the number of markers in the file, use SONMarkerGetCount. [☞ SONMarkerGetCount, p. 357.](#)

The channel must be an RealMark channel with a value array of size one. [☞ SONChannelGetType, p. 353.](#)
[☞ SONMarkerGetValueCount, p. 362.](#)

11.5.25 SONGetStatusText

SONGetStatusText translates a status value returned by a DataAccess operation to a text string.

Syntax

SONGetStatusText *session, status, message*

session A Long value representing the current session.

status A Long value representing the status value to translate.

message A Variant variable in which the translated text is returned.

Example

```
Dim status As Long
Dim message As Variant
Dim file_name As Variant
file_name = "Sample.dat"
```

```
status = SONFileOpen(session, file_name)
```

```
If status <> 0 Then
    SONGetStatusText session, status, message
    ' Display the message here
End If
```

Discussion

SONGetStatusText does not return a value, so it should not be called as a function.

You need not have a file open.

SONGetStatusText is the only means provided to interpret status values.

11.5.26 SONGetVersion

SONGetVersion returns the version number of the DataAccess package.

Syntax

SONGetVersion *version*

version A Long variable to receive the DataAccess version number.

Example

```
Dim version As Long
```

```
SONGetVersion version
```

Discussion

SONGetVersion does not return a value, so it should not be called as a function.

You need not have either a session or a file open.

If you are writing a set of procedures using DataAccess, you can use SONGetVersion to ensure that the correct version is being used.

For example, suppose that your procedures require at least version N of the DataAccess. You can use the following code to ensure that this version or a later version is in use:

```
Dim version As Long
SONGetVersion version
```

```
If version < N Then
    ' Handle the error
End If
```

11.5.27 SONSessionClose

SONSessionClose ends the session. It deallocates any resources allocated DataAccess.

Syntax

```
SONSessionClose session
```

session A Long value representing the current session.

Example

```
SONFileClose session
SONSessionClose session
```

Discussion

SONSessionClose does not return a value, so it should not be called as a function.

You should close a session to deallocate any resources allocated to the session.

11.5.28 SONSessionOpen

SONSessionOpen begins a DataAccess session.

Syntax

```
SONSessionOpen session
```

session A Long variable to contain the created session.

Example

```
Dim session As Long
Dim result As Long
Dim file_name As Variant
file_name = "Sample.dat"

SONSessionOpen session

result = SONFileOpen(session, file_name)
```

Discussion

You must have a session open to use any other DataAccess calls. The handle returned by SONSessionOpen is a parameter to all other calls.

You can have as many sessions open simultaneously as you would like. For example, if you want to have two data files open simultaneously, call SONSessionOpen twice. Use one handle when accessing one file and the other handle when accessing the other file.

After you are done with the session, you should call SONSessionClose. [☞ SONSessionClose, p. 364.](#)

11.5.29 SONSegmentGetCount

SONSegmentGetCount returns the number of segments recorded for the channel.

Syntax

SONSegmentGetCount *session, channel, count*

session A Long value representing the current session.

channel A Long value specifying the channel.

count A Long variable in which to store the number of segments.

Example

```
Dim segment_count As Long
```

```
SONSegmentGetCount session, segment_count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The channel must be an Adc or RealWave channel. [☞ SONChannelGetType, p. 353.](#)

The segment count is the number of segments recorded in the SON data file. Segments are numbered 0 to N-1, where N is the number of segments in the file.

11.5.30 SONSegmentGetRange

SONSegmentGetRange returns the range of the specified segment.

Syntax

SONSegmentGetRange *session, channel, segment, start, sample_count*

session A Long value representing the current session.

channel A Long value specifying the channel.

start A Double variable in which to return the segment start time.

segment A Long value specifying the segment.

sample_count A Long variable in which to return the segment sample count.

Example

```
Dim start As Double
```

```
Dim sample_count As Long
```

```
SONSegmentGetRange session, 0, 0, start, sample_count
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. [☞ SONChannelGetCount, p. 351.](#)

The channel must be an Adc or RealWave channel. [☞ SONChannelGetType, p. 353.](#)

Segments are numbered 0 to N-1, where N is the number of segments recorded in the file. To determine the number of segments in the file, use SONSegmentGetCount. [☞ SONSegmentGetCount, p. 365.](#)

The start of a sweep is measured in seconds from the file creation time.

11.5.31 SONSegmentRead

SONSegmentRead reads the data for the specified channel and segment.

Syntax

```
SONSegmentRead
    session, channel, segment, start, length, data
```

session A Long value representing the current session.

channel A Long value specifying the channel.

segment A Long value specifying the segment.

start A Long value representing the first sample to read.

length A Long value representing the number of samples to read.

data An array of "Real" to receive the data. The array must be long enough to contain all the data in the section.

Example

```
Dim samples As Long
Dim start As Long
Dim buffer() As Real
```

```
SONSegmentGetSampleCount session, 0, 0, samples
```

```
samples = samples/2
start = samples/2
```

```
ReDim buffer(samples)
```

```
SONSegmentRead
    session, 0, 0, start, samples, buffer(0)
```

Discussion

You must have a file open.

Channels are numbered 0 to N-1, where N is the number of channels recorded in the file. To determine the number of channels in the file, use SONChannelGetCount. *☞ SONChannelGetCount, p. 351.*

The channel must be an Adc or RealWave channel. *☞ SONChannelGetType, p. 353.*

Segments are numbered 0 to N-1, where N is the number of segments recorded in the file. To determine the number of segments in the file, use SONSegmentGetCount. *☞ SONSegmentGetCount, p. 365.*

The start and length values must satisfy the following criteria:

- 1 The start value must not be negative.
- 2 The length value must be greater than zero.