

Bachelor Thesis

Spektrometer App

Anbindung Spektrometer an mobiles Device

Andreas Lüscher & Raphael Bolliger



Fachhochschule Nordwestschweiz
Hochschule für Technik

Dozent: Martin Gwerder
Auftraggeber: Andreas Hueni

Windisch, 23. März 2017

Inhaltsverzeichnis

1	Einleitung	1
1.1	Ziel der Arbeit	1
1.2	Hilfestellungen	1
1.3	Erreichtes	1
1.4	Leserführung	2
2	Theoretischer Teil	3
2.1	Einleitung	3
2.2	Messverfahren	3
2.2.1	Abspeichern vs. Anzeigen	3
2.3	Berechnungsablauf	4
2.4	Berechnungen	4
2.4.1	Dark Current	4
2.4.2	White Reference	5
2.4.3	Radiance	5
3	Umsetzung	6
3.1	Einleitung	6
3.2	Technologien	6
3.2.1	Entwicklungsumgebung	6
3.2.2	Programmiersprache	6
3.2.3	Systemgrenzen	6
3.2.4	Abhängigkeiten	7
3.2.5	Deployment	8
3.3	Architektur	9
3.3.1	Grobarchitektur	9
3.3.2	Core	9
3.3.3	Service	9
3.3.4	iOS App	10
3.4	Core	10
3.4.1	Models	10
3.4.1.1	Command	10
3.4.1.2	SpectralData	10
3.4.1.3	IndicoFile	10
3.4.2	Connections	10
3.4.3	Input/Output	11
3.4.3.1	File Writer	11
3.4.3.2	File Reader und Spectrometer Parser	11
3.4.4	Calculations	11
3.4.5	Error Handling	12
3.5	Service	13
3.5.1	Instrument Store	13

3.5.2	Command Manager	13
3.5.3	File Write Manager	13
3.6	iOS App	14
3.6.1	App Delegate	14
3.6.2	Core Data	14
3.6.3	Views	14
3.6.4	Controllers	15
3.6.4.1	Connections	15
3.6.4.2	Spectrometer	15
3.6.4.3	MeasurementSeries	15
3.6.4.4	Measurements	17
3.6.4.5	Settings	18
3.6.5	Components	18
3.6.6	View Store	18
3.6.7	Service	19
3.6.7.1	Validation	19
3.6.7.2	File Browser	19
4	Testing	21
4.1	Unit Tests	21
4.1.1	IO Tests	21
4.2	Integration Tests	21
5	Projektorganisation	22
5.1	Vorgehen	22
5.2	Risiken	22
5.2.1	ASD Dokumentation stimmt nicht mit Realität überein	22
5.2.2	Zu wenig Zeit um alle Anforderungen zu erfüllen	22
5.2.3	Daten vom iOS Gerät importieren und exportieren	23
5.3	Meilensteine	24
5.4	Zeitplanung	24
5.5	Anforderungen	24
5.6	Change Management	25
5.7	Arbeitspakete	25
5.8	Soll- und Istvergleich	26
6	Fazit	27
6.1	Zusammenfassung	27
6.2	Potentielle Erweiterungen	27
6.2.1	Neues ASD Gerät	27
6.2.2	Absolut Reflectance	27
6.2.3	GPS Daten und Fotos zu Messungen hinzufügen	27
6.3	Erkenntnisse	28
7	Ehrlichkeitserklärung	29
8	Abbildungsverzeichnis	30
9	Glossar	31
10	Literaturverzeichnis	32

11 Anhang	33
11.1 Zeitplan	33
11.2 Risikoanalyse	33
11.3 Testprotokolle	33
11.4 ASD	33
11.4.1 Indico File Format	33
11.4.2 ASD Developers Guide	33

Zusammenfassung

Das Hauptziel des Projektes ist, eine mobile Applikation zu erstellen, welche die RS³ Desktoplösung von ASD ablöst. RS³ ist eine Software, welche die Verbindung zu einem Spektralmessgerät herstellt, Messungen auslösen sowie die Resultate anzeigen kann. Das bisherige System besteht aus einem Laptop inklusive RS³ Software, welche jeweils auf ein Spektrometer abgestimmt ist. Neu soll eine mobile App ausreichen, um mehrere Spektrometer ansprechen zu können. Die App soll Forschende unterstützen, Messungen direkt im Feld zu beurteilen und verwalten zu können. Die erstellten Messungen sollen abschliessend exportiert werden können, um diese am Computer weiterzuverwenden.

1 Einleitung

1.1 Ziel der Arbeit

Das geologische Institut der Universität Zürich betreibt zur Forschung vier Spektrometer der Firma ASD Inc. aus Colorado in den USA. Zu jedem Spektrometer liefert ASD ein Notebook mit installierter Software, um das [Spektrometer](#) zu bedienen und Messungen auszuführen.

Ziel dieser Arbeit war, die Software [RS³](#) von ASD mit einer modernen Applikation für ein mobiles Device abzulösen. Das Projektteam hat sich gemeinsam mit dem Kunden dazu entschieden, die Applikation für iOS Geräte, im speziellen iPads, zu entwickeln.

1.2 Hilfestellungen

Zur Umsetzung konnten verschiedene Hilfestellungen in Anspruch genommen werden. ASD bietet auf ihrer Webseite einen [Download](#) mit einem Developer-Kit an. In dieser Dokumentation ist beschrieben, wie interessierte Entwickler mittels eines TCP-Servers, der auf dem Spektrometer betrieben wird, selbst Applikationen entwickeln können. Die Dokumentation enthält ausführliche Informationen zu Verbindungsparameter, Rückgabetypen und Dateiformaten.

Weiter konnte auf das [GitHub Repository](#) der SPECCHIO-Datenbank zurückgegriffen werden. In dieser Applikation wurden verschiedenste Berechnungen und Manipulationen mit Spektraldateien bereits in Java programmiert.

1.3 Erreichtes

Die Applikation wurde mit den definierten Grundanforderungen vollständig umgesetzt. Der Benutzer kann, sofern das iPad mit dem Spektrometer über WLAN verbunden ist, das Spektrometer bedienen und Messungen ausführen. Es wurde speziell darauf geachtet, den Messablauf einfacher und für den Benutzer intuitiver zu gestalten. Die Grundansicht wurde nahezu von der bestehenden Software übernommen. Somit sollte es für die Benutzer keine zu grosse Umstellung sein.

Weiter wurde darauf geachtet, die Applikation in Zukunft noch weiter zu entwickeln. Die Architektur wurde stark strukturiert, damit auch Personen, die noch nicht mit dem Projekt vertraut sind, eine Weiterentwicklung vornehmen können.

1.4 Leserführung

Dieses Dokument beschreibt die Erarbeitung eines Informatik Projektes 6 der Fachhochschule Nordwestschweiz. Die Dokumentation ist in einen theoretischen und einen praktischen Teil aufgeteilt. Im theoretischen Teil wird nur kurz erklärt was ein Spektrometer genau misst und wie die Daten berechnet und abgespeichert werden. Im praktischen Teil wird vor allem die Softwarearchitektur und die konkrete Umsetzung genau beschrieben.

2 Theoretischer Teil

2.1 Einleitung

Ein Spektrum besteht aus 2051 Zahlenwerten. Im Fall der FieldSpec 3 und 4 Spektrometer wird dieses Spektrum noch in drei Teile unterteilt. [VNIR](#), [SWIR1](#) und [SWIR2](#).

2.2 Messverfahren

Das Spektrometer misst die Werte immer im gleichen Verfahren. Als Rückgabe vom Gerät wird immer die gleiche Datenstruktur verarbeitet. Diese ist unter dem Abschnitt [3.4.3.2](#) genauer beschrieben. Damit später die Berechnungen angewendet und die Messwerte in Dateien abgespeichert werden können, müssen einige Referenz-Werte vorgängig vorhanden sein. Diese werden zum Teil ebenfalls direkt mit dem Spektrometer erfasst, wie eine White Reference oder Dark Current. Referenz-Werte für die Radiance-Berechnung sprich die Base, Lamp und FiberOptic Dateien, werden beim Konfigurieren des Spektrometers in die App importiert und abgespeichert.

2.2.1 Abspeichern vs. Anzeigen

Die berechneten Daten der Reflectance und Radiance werden in der App **nur** angezeigt aber niemals in eine Datei geschrieben. Die Daten die in den Messdateien abgespeichert werden sind immer Raw-Daten. In den Messdateien sind somit alle Daten vorhanden um später wieder eine Reflectance oder Radiance Berechnung durchzuführen.

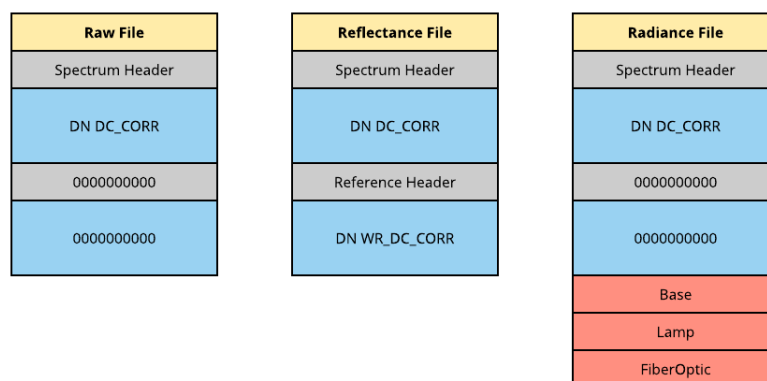


Abbildung 2.1: Inhalt der Messdateien

2.3 Berechnungsablauf

Im untenstehenden Diagramm wird der Messablauf grafisch dargestellt. An der blauen Linie ist zu erkennen, dass wiederum nur **DN DC_CORR** und **DN WR_DC_CORR** in die Messdateien geschrieben werden. Die berechneten Werte von Reflectance und Radiance werden nur für die Diagramm-Anzeige im App verwendet. Auf die genauen Berechnungen wird im nächsten Kapitel genauer hingewiesen.

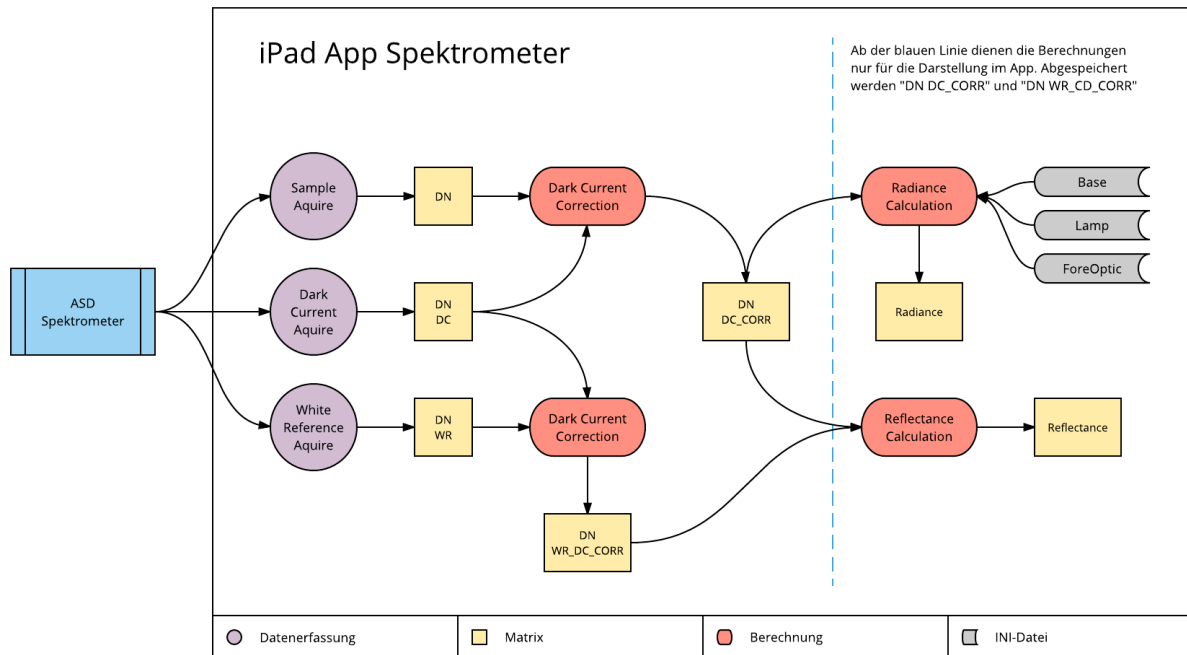


Abbildung 2.2: Berechnungen für Raw, Reflectance und Radiance

2.4 Berechnungen

2.4.1 Dark Current

Als Dark Current wird das Spektrum bezeichnet, das gemessen wurde, nachdem dem Spektrometer komplett das Licht genommen wurde. Dies erreicht man auf unterschiedliche Weise. Eine Möglichkeit ist das Kappen der **Faser** oder das Schliessen des mechanischen Shutter. Es gibt bei neueren Geräten eine weitere Möglichkeit die Dark Correction mit einer im voraus geladenen Konfigurationsdatei auszuführen. Im aktuellen Projekt wird nur die Variante mit dem mechanischen Shutter unterstützt.

Die Berechnung wird nur auf den **VNIR** Bereich des Spektrums angewendet.

$$\forall i \in \{0, \dots, n\} DC_s(i) = Ts(i) - Ds(i) + (V_{\text{DarkCurrentCorrection}} + (T_{\text{drift}} - D_{\text{drift}}))$$

n = size of the VNIR spectrum

DC_s = dark corrected spectrum

T_s = current measured spectrum

D_s = dark measured spectrum

$V_{\text{DarkCurrentCorrection}}$ = dark current correction constant¹

T_{drift} = current measured drift value

D_{drift} = dark measured drift value

2.4.2 White Reference

Als White Reference wird das Spektrum bezeichnet, welches aus der Division eines normalen Spektrums durch eines über einer Weissreferenzplatte erfassten Spektrums entsteht. Die Berechnung wird bei der White Reference auf das gesamte Spektrum angewendet.

$$\forall i \in \{i, \dots, n\} DW R_s(i) = \frac{T_s(i)}{W_s(i)}$$

n = size of the complete spectrum

$DW R_s$ = calculated white reference spectrum

T_s = current measured spectrum

W_s = reference measured spectrum

2.4.3 Radiance

Für die Berechnung der Radiance sind nebst dem aktuellen Spektrum auch die Referenzwerte aus den Base, Lamp und ForeOptic Dateien notwendig. Diese werden direkt beim Konfigurieren eines Spektrometers angehängt. Ein Spektrometer kann auch ohne ForeOptic Dateien konfiguriert werden, eine Radiance Berechnung ist in diesem Falle dann aber nicht möglich.

Die Berechnung der Radiance wurde vom Auftraggeber übernommen. Das Referenzprojekt ist auf Github vorhanden. Der Link auf das File lautet:

https://github.com/ahueni/SPECCHIO/.../reader/spectrum/ASD_FileFormat_V7_FileLoader.java

Die erwähnte Radianzberechnung befindet sich in der Methode `convert_DN2L`.

¹Dieser Wert wird beim Verbindungsaufbau mit dem Spektrometer ausgelesen und im App zwischengespeichert.

3 Umsetzung

3.1 Einleitung

Dieses Kapitel beschreibt die Umsetzung des Produktes. Die nachfolgenden Abschnitte werden schrittweise detaillierter und beschreiben den Aufbau sowie die Designentscheidungen des Softwarecodes. Es soll eine zukünftige Weiterentwicklung der Software ermöglichen und für den Entwickler vereinfachen.

3.2 Technologien

In diesem Abschnitt wird beschrieben, welche Technologien und Tools eingesetzt wurden, um die iOS Applikation umzusetzen. Es wurde darauf geachtet, dass bewährte und von Apple vorgesehene Technologien verwendet werden. Die Applikation sowie der Code entsprechen der von Apple vorgegebenen Richtlinien. Weitere Informationen für die Entwicklung von Applikationen für die iOS Betriebssysteme sind direkt im [Apple Entwicklungsbereich](#) zu finden.

3.2.1 Entwicklungsumgebung

Das Projekt wurde mit XCode 8.2.1 umgesetzt. Grundsätzlich kann jede [IDE](#) verwendet werden, welche mit XCode-Projektdateien kompatibel ist. Die einzige Voraussetzung, um die Applikation erfolgreich zu kompilieren, ist ein Computer mit macOS Betriebssystem.

Es können später weitere Voraussetzungen dazukommen, je nachdem wie die Applikation an die tatsächlichen Benutzer ausgeliefert werden soll.

3.2.2 Programmiersprache

Das Projekt wurde in der Programmiersprache Swift 3 umgesetzt und ist mit iOS 10.0 und höher kompatibel. Es wurde darauf geachtet, dass alle Abhängigkeiten ebenfalls in Swift umgesetzt sind. Dies erleichtert die Weiterentwicklung, da kein, für ungeübte Entwickler, meist schwer lesbarer Objective-C Code zum Einsatz kommt.

3.2.3 Systemgrenzen

Da das Projekt für die iOS Plattform entwickelt wird, werden einige Funktionen bereits vom System übernommen. Folgende Funktionen werden ganz oder teilweise vom System bereitgestellt:

- Import / Export Die Import sowie die Export Funktionen werden zu einem Grossteil vom System übernommen.
- Core Data Objekte werden vom System verwaltet

Alle anderen Komponenten, wurden im Projekt entwickelt oder ins Projekt inkludiert.

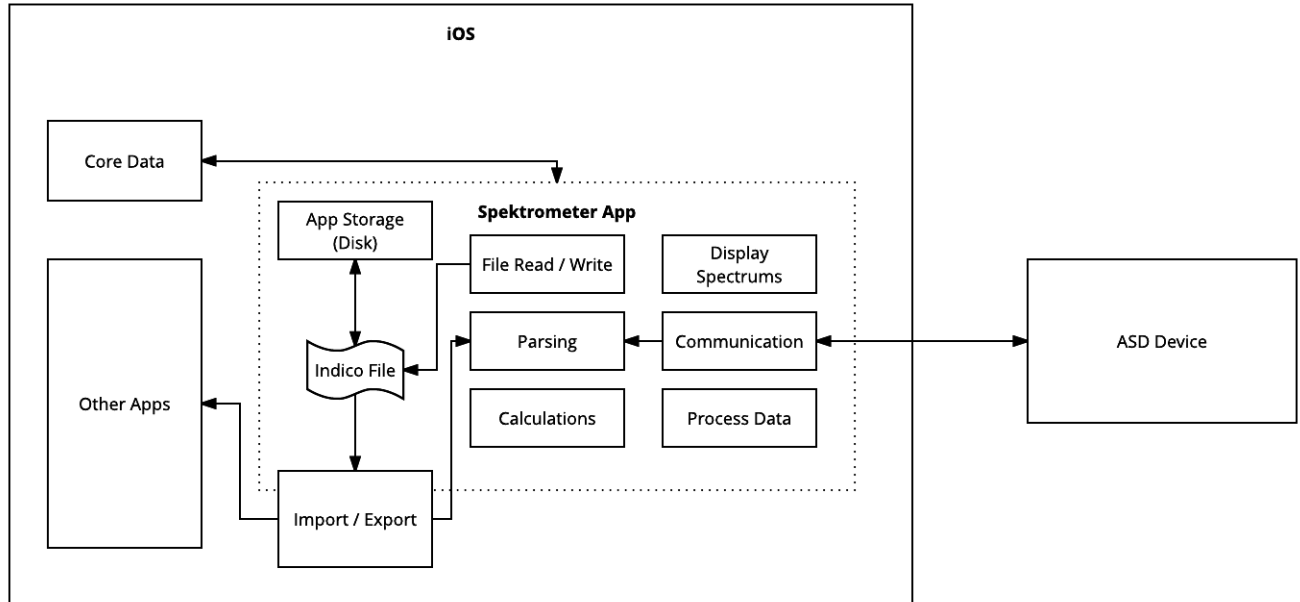


Abbildung 3.1: Systemgrenzen

3.2.4 Abhängigkeiten

Externe Ressourcen und Frameworks, wurden vorwiegend mit CocoaPods eingebunden. CocoaPods ist ein Paket Manager, mit welchem man Pakete in ein bestehendes Projekt einbinden kann. Dies geschieht über ein sogenanntes Podfile, welches zum Projektumfang gehört. Dieses File beschreibt die externen Abhängigkeiten, sowie welche Version benötigt wird. CocoaPods ist kein Produkt von Apple und wird durch die Community weiterentwickelt und bereitgestellt. Sollten Updates zu Frameworks bereitstehen, können diese einfach ins Projekt integriert werden.

Folgende Frameworks wurden mit CocoaPods in das Projekt eingebunden und verwendet:

- Charts: <https://github.com/danielgindi/Charts>
- Zip: <https://github.com/marmelroy/Zip>
- PlainPing: <https://github.com/naptics/PlainPing>
- Pulsator: <https://github.com/shu223/Pulsator>
- FontAwesome.swift: <https://github.com/thii/FontAwesome.swift>

Folgende Frameworks wurden direkt als Klassendateien ins Projekt eingebunden und verwendet, da sie nicht als Paket verfügbar sind.

- Socks: <https://github.com/vapor/socks>

Das folgende Framework wurde teilweise übernommen und zusätzlich modifiziert. Es wurde auf die projektspezifischen Eigenschaften reduziert und angepasst.

- iOS FileBrowser in Swift: <https://github.com/marmelroy/FileBrowser>

3.2.5 Deployment

Die Applikation wurde während der Entwicklung mit dem von Apple zur Verfügung gestellten Service [TestFlight](#) ausgeliefert. Somit konnten die Prototypen und die Vorabversion auf einfache Art und Weise dem Kunden zur Verfügung gestellt werden.

[TestFlight](#) ermöglicht es Prototypen und Vorversionen über einen gesonderten AppStore auf registrierten iOS Geräten von bis zu 200 Testpersonen als App-Download anzubieten. Um diesen Dienst zu nutzen ist ein Apple Developer Account notwendig, mit der Integration und Konfiguration von zusätzlichen Zertifikaten.

Für eine zukünftige Weiterentwicklung und vor allem für die Auslieferung an externe Benutzer, kann die Applikation ebenfalls in den AppStore aufgenommen werden oder als B2B¹ Applikation vertrieben werden. In beiden Fällen benötigt der Entwickler aber einen gültigen Apple Developer Account.

¹Develop Custom Apps for Business: <https://developer.apple.com/programs/volume/b2b/>

3.3 Architektur

3.3.1 Grobarchitektur

Um weite Teile des Quellcodes erneut nutzen zu können, wurde das Projekt in die drei Teile Core, Service und iOS App aufgeteilt. Die Layer wurden nach bestimmten Kriterien aufgetrennt, diese werden in den nachfolgenden Kapiteln genau beschrieben.

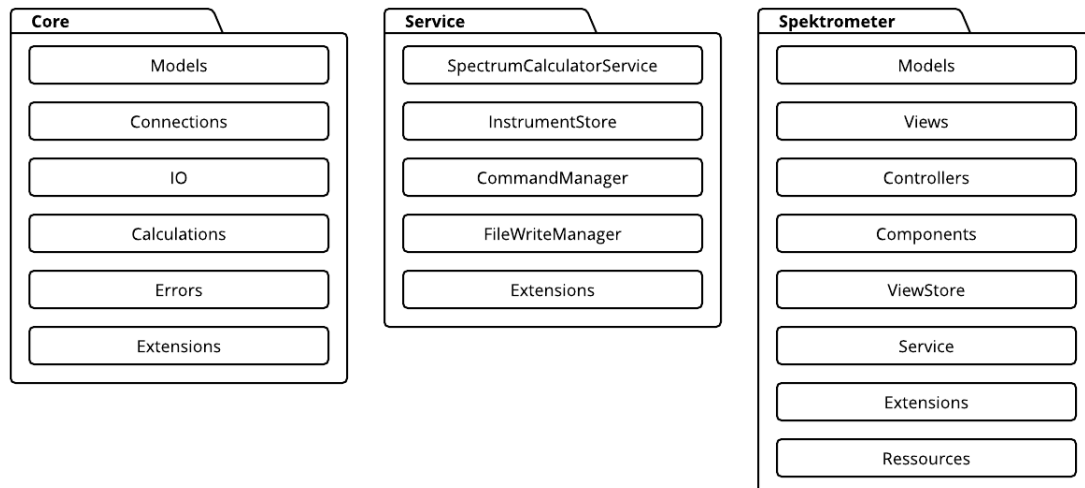


Abbildung 3.2: Layeraufteilung der Grobarchitektur

3.3.2 Core

Der Core fasst den gesamten Code zusammen welcher plattformunabhängig ist. Dieser Code könnte in weiteren Projekten eingesetzt werden, die auf der Programmiersprache Swift aufbauen. Klassen im Core-Layer verwenden nur Grundkomponenten von Swift und könnten so auch auf einem anderen Betriebssystem eingesetzt werden, das Swift unterstützt.

3.3.3 Service

Der Service-Layer kapselt Anfragen des iOS-Layers an den Core. Dieser Layer wurde verwendet um Parallelität zu vermeiden. Der Service-Layer beinhaltet Manager-Klassen, die Aufrufe an den Core mit einer Synchronized-Queue bearbeiten und weiterleiten. Somit ist garantiert, dass niemals zwei Befehle gleichzeitig oder überlappend an das [Spektrometer](#) gesendet werden. Auch das Schreiben von nummerierten Messdateien ins Dateisystem kann durch ein FileManager so gekapselt werden.

Klassen im Service-Layer unterscheiden sich durch ein weiteres Kriterium vom Core. Sie können nur auf Plattformen von Apple eingesetzt werden, da sie gewisse Importe benötigen die nicht in der Swift Basis enthalten sind. Der Service-Layer könnte im gesamten Programmierumfeld von Apple (macOS, iOS, watchOS, tvOS) eingesetzt werden, nicht aber ausserhalb.

3.3.4 iOS App

Dieser Layer (Im Xcode wird dieser Ordner als Spectrometer geführt) beinhaltet die eigentliche iOS App mit all ihren typischen Eigenschaften wie das AppDelegate, die Storyboards und die ViewController. Zusätzlich wurden noch eigene View Komponenten erstellt die ebenfalls in diesem Layer enthalten sind.

3.4 Core

3.4.1 Models

In diesem Ordner werden sämtliche Datenstrukturen definiert die in Verbindung mit dem Spektrometer, den INI-Dateien oder Indico-Dateien verwendet werden. Es wurde darauf geachtet, diese möglichst erweiterbar zu gestalten. Es gibt wenn möglich eine Basis-Klasse, welche die immer gleichen Grundstrukturen bei Spektraldaten oder Indico-Dateien implementiert. Die Ableitungen spezialisieren dann die effektiven Datentypen. Diese Aufteilung ermöglicht es einem Entwickler, einfach einen neuen Indico-Standart oder ein Rückgabetyt eines zusätzlichen Spektrometers zu erstellen.

3.4.1.1 Command

Der Command wird verwendet, um genormte Anfragen ans Spektrometer zu senden. Für jeden Command wurde eine Grösse definiert, die vom erwarteten Datentyp abhängt. Somit kann der TCP Manager jeweils entscheiden, wann alle Daten empfangen wurden.

3.4.1.2 SpectralData

In diesem Ordner wurden alle Datentypen definiert, die vom Spektrometer empfangen werden können. Die genauen Grössen und vorhandenen Felder wurden dem ASD Developer Guide entnommen. Dieser ist im [Anhang](#) erwähnt. Die meisten Befehle haben ihren bestimmten Rückgabetyt. Einige verwandte Befehle verwenden jedoch auch den gleichen Rückgabetyt.

3.4.1.3 IndicoFile

In diesem Ordner wurde der Datentyp für das Indico-Dateiformat definiert. Die Basisklasse `IndicoFileBase` repräsentiert genau eine INI-Datei, die für die Spektrometer-Konfiguration verwendet wird. Die Ableitung `IndicoFile7` repräsentiert die vollständige Datei mit allen Spektraldaten. Der Inhalt der Dateien wird in der Abbildung [2.1](#) genau dargestellt.

3.4.2 Connections

In diesem Ordner, befindet sich die TCP-Manager Klasse sowie die externen Socks Dateien. Das Socks Framework kümmert sich um die Kommunikation auf TCP Ebene. Für die Integration wurde

nicht CocoaPods verwendet, weshalb die Klassen direkt in den Ordner kopiert wurden.

Der TCP-Manager ist für die Anfragen an das Spektrometer verantwortlich. Es ist die einzige Klasse, welche direkt mit dem [Spektrometer](#) kommuniziert und ist deshalb als Singleton umgesetzt. Mit der Methode `connect(internetAddress: InetAddress)` kann eine Verbindung zum [Spektrometer](#) hergestellt werden, danach können beliebig viele Kommandos mit `sendCommand(command: Command)` gesendet werden. Um die Verbindung zu schliessen, muss lediglich die `disconnect()` Methode aufgerufen werden.

3.4.3 Input/Output

In diesem Kapitel werden alle Klassen beschrieben, welche direkt Dateien einlesen, Daten empfangen oder Dateien schreiben.

3.4.3.1 File Writer

Die Output Klassen dienen dazu, eine Indico-Datei zu schreiben. Die Basis Klasse kapselt dafür alle Schreibvorgänge. Von der Klasse `FileWriter` wird auf diese Methoden zugegriffen und stösst so das Schreiben in der richtigen Reihenfolge an. Von aussen kann lediglich gewählt werden, welche der drei Messmethoden geschrieben werden soll: Raw, Reflectance oder Radiance. Um die Applikation um einen neuen Writer zu erweitern, muss lediglich vom `BaseWriter` abgeleitet werden und eine eigene `write()` Methode implementiert werden. Diese neue Klasse kann die von der Basis bereitgestellten Methoden nutzen oder um weitere ergänzen.

3.4.3.2 File Reader und Spectrometer Parser

Die Reader- und Parserklassen bauen alle auf der selben Basis auf. In der `BaseSpectrumInput` Klasse sind alle Funktionen realisiert, um aus Byte-Werten die entsprechenden Datentypen zu parsen. In den konkreten Implementierungen müssen diese dann nur in der richtigen Reihenfolge aufgerufen und den entsprechenden Feldern zugewiesen werden.

Durch die inkonsistente Implementierung von der Firma ASD gibt es immer wieder verwirrende Stellen die gesondert implementiert werden müssen. Als Beispiele sind hier Strings zu nennen, die zum Teil als prefixed, null-terminiert oder aber mit fixer Länge in der gleichen Datei implementiert wurden. Teilweise werden Float- oder Doublewerte in derselben Antwort vom Gerät einmal im "Little-Endian"-und dann wieder im "Big-Endian"-Format codiert.

3.4.4 Calculations

Der Ordner Calculations enthält lediglich die Klasse `SpectrumCalculator` welche nur statische Funktionen enthält. Diese dienen der Berechnung der Reflektanz sowie der Radianz. Ausserdem beinhaltet die Klasse noch eine Methode um eine Dark Correction auf einem übergebenen Spektrum durchzuführen.

3.4.5 Error Handling

Fehler werden nur abgefangen wenn sie erwartet werden. Dies bedeutet es gibt kein allgemeine Fehlerbehandlung, da dies die Applikation in einen inkonsistenten Zustand bringen würde. Das ErrorHandler wurde nur eingebaut bei der Kommunikation mit externen Ressourcen, sprich dem ASD Gerät oder bei Input/Output Operationen. Bei der Entwicklung, wurde festgestellt, dass die Daten des Spektrometers nicht zwingend korrekt sein müssen. In diesem Fall, wird beim Parsen ein `SpektrometerError` geworfen. Hier am Beispiel, des Saturation Enums:

```
enum Saturation: Int {
    case NoSaturation = 0
    case Saturated = 1
    init(fromRawValue: Int) throws {
        if let value = Saturation(rawValue: fromRawValue) {
            self = value
        } else {
            throw SpectrometerError(message: "Could not parse saturation
            of spectrum.", kind: .parsingError)
        }
    }
}
```

Das `SpectrometerError` Objekt sieht wie folgt aus:

```
struct SpectrometerError: Error {
    enum ErrorKind {
        case connectionError
        case readError
        case parsingError
    }
    let message: String
    let kind: ErrorKind
}
```

Der `CommandManager` ist zuständig diese Errors korrekt zu behandeln. Dies geschieht, indem er sie bei privaten Methoden weitergibt und bei öffentlichen Funktionen abfängt. Jede öffentliche Methode des `CommandManager` besitzt zwei Callback Parameter. Jeweils ein Callback für den erfolgreichen durchlauf der Methode und ein Fehler Callback. Wird ein Fehler abgefangen, wird der Fehler Callback aufgerufen, ansonsten der success Callback. Folgend die aquire Methode als Beispiel:

```
func aquire(samples, successCallback:(spectrum), errorCallback:(error)) {
    var spectrum: FullRangeInterpolatedSpectrum!
    do {
        try serialQueue.sync {
            spectrum = try internalAquire(samples: samples)
        }
        successCallback(spectrum)
    } catch let error {
        errorCallback(error)
    }
}
```

Nun kann der Aufrufer entscheiden, was er bei einem Fehler oder einem korrekten Aufruf machen möchte. Dies geschieht in der Regel in der ViewController Methode. Falls beim Parsen ein Fehler auftritt, wechselt die Applikation auf die Verbindungsvue und trennt die TCP Verbindung zum Spektrometer. So ist sichergestellt, dass keine falschen Daten verarbeitet oder angezeigt werden.

3.5 Service

3.5.1 Instrument Store

Im `InstrumentStore` werden alle Objekte gespeichert, auf welche global zugegriffen werden muss und die nichts mit der Benutzeroberfläche zu tun haben. Um zu garantieren, dass die Variablen nur einmal existieren, ist die Klasse als Singleton umgesetzt.

3.5.2 Command Manager

Der `CommandManager` übernimmt die Funktion, alle Kommandos in der korrekten Reihenfolge abzuarbeiten. Der `CommandManager` bietet Methoden für alle gängigen `Spektrometer` Kommandos an. Wird eine dieser Methoden aufgerufen, wird die Anfrage an eine synchrone Queue übergeben welche nach dem First-In-First-Out Prinzip die Anfragen abarbeitet. So ist sichergestellt, dass ein Kommando erst an das `Spektrometer` gesendet wird, wenn alle vorherigen abgearbeitet sind. Der `CommandManager` besitzt eine Methode, um einen Callback zu erhalten, sobald dieser an der Reihe ist. Dies wird benötigt, um nach dem beenden des AquireLoops feststellen zu können, wann der letzte Command in der Queue abgearbeitet wurde. Die Methode sieht wie folgt aus:

```
func addCancelCallback(message: String? = nil, callBack: () -> Void) {
    serialQueue.sync {
        callBack()
    }
}
```

3.5.3 File Write Manager

Der `FileWriteManager` funktioniert ähnlich wie der `CommandManager`. Er übergibt dem `FileWriter` Aufträge ebenfalls in einer synchronen Warteschlange und arbeitet diese ab. Dazu kümmert sich der `FileWriteManager` auch um die korrekte Benennung der ASD Files. Dies geschieht, indem er den höchsten geschriebenen prefix eines Files im Pfad mit dem selben Basisnamen findet und diesen dann entsprechend erhöht.

Der `FileWriteManager` verfügt wie der `CommandManager` über eine Callback Methode, welche die Callback Methode aufruft, sobald das letzte File in der Queue geschrieben wurde.

3.6 iOS App

3.6.1 App Delegate

Das `AppDelegate` ist der Einstiegspunkt jeder iOS Applikation. Darin wird beschrieben, wie der Prozess aus verschiedenen Zuständen abläuft (Stopped, Paused, Notified). Im `AppDelegate` können auch applikationsweite Designanpassungen vorgenommen werden. So kann eine Schriftart für `TabBarItem`s oder für die `NavigationBar` definiert werden.

Das `AppDelegate` wird ebenfalls aufgerufen, wenn eine externe Datei erfolgreich importiert wurde. Das Importieren selbst wird vom Betriebssystem übernommen. In diesem Projekt wird auf der aktuellen View eine Meldung angezeigt, dass eine Datei erfolgreich importiert wurde.



Vorsicht

Die Importfunktion funktioniert beim importieren aus iCloud Drive nicht. Apple übergibt nur die URL der Datei aus dem iCloud Drive App. Das Spektrometer App hat kein Zugriff auf diese Datei.

3.6.2 Core Data

Core Data ist ein System, um Daten auf dem Gerät zu speichern. Es dient nicht als Datenbanksatz, kann aber für einfache Speicherungen genutzt werden. Durch die Integration in XCode, lassen sich Speicherklassen einfach modellieren und einsetzen. Core Data Einstellungen können in `xcdatamodeld` Klassen vorgenommen werden. Dabei können Datentypen sowie die Relationen der einzelnen Records definiert werden. Weitere Informationen befinden sich auf der offiziellen Entwicklerseite von Apple im [Core Data Programming Guide](#)



Vorsicht

Core Data Zugriffe sind vergleichsweise langsam und sollten daher nicht in Schleifen verwendet werden.

3.6.3 Views

Alle ViewController sind in der Datei `Main.storyboard` enthalten. Einzig das Design für zwei TableView-Zellen wurde in separate XIB-Dateien ausgelagert. Dies war eine bewusste Entscheidung, um Entwicklern einen guten Überblick über den gesamten Ablauf in der Benutzeroberfläche zu ermöglichen. Ein ViewController wird von iOS eigenhändig instanziiert, sobald er von einer anderen View aufgerufen wird. Dies ermöglicht es, Verlinkungen der Views direkt im Storyboard umzusetzen.

Die Anordnung der Controls wurde im Storyboard gelöst. Kleinere Merkmale wurden jeweils im Code angepasst, indem diese direkt beim Laden eines ViewControllers vorgenommen wurden.

Die gesamte Ausrichtung und Anpassung auf unterschiedliche Bildschirmgrößen wurde ebenfalls im Storyboard mit **Auto Layout** gelöst. Einzige Ausnahme ist der **SpectrometerViewController**. **Auto Layout** ermöglicht keine Unterscheidung des Portrait oder Landscape Modus bei iPads. Diese Unterscheidung und das Verhalten wurden direkt im Code umgesetzt.

3.6.4 Controllers

3.6.4.1 Connections

Die zwei ViewControllers **AddEditConnectionViewController** und **ConnectionViewController** dienen beim Start der Applikation, dazu neue Spektrometer hinzuzufügen oder bestehende zu bearbeiten. Diese ViewController interagieren direkt mit dem Storyboard und den dazugehörigen Views. Die Klasse **SpectrometerConfigTableViewCell** dient als Hilfsklasse, um die Tabelleneinträge der Spektrometertabelle zu gestalten und mit dem Code zu verknüpfen.

3.6.4.2 Spectrometer

In der Klasse **SpectrometerViewController** sind alle Funktionen und Verknüpfungen für die Hauptansicht umgesetzt. Diese ist in der Lage, wenn alle Voraussetzungen erfüllt sind, zwischen Raw, Reflectance und Radiance zu wechseln und die berechneten Daten direkt im Liniendiagramm darzustellen. Das kontinuierliche akquirieren neuer Messwerte wird durch eine Endlosschleife immer weitergeführt bis die Abbruchbedingung eintritt. Weiter ist die Klasse für folgende Aktionen und Darstellungen zuständig:

- Dark Current oder White Reference per Button auslösen.
- Die entsprechenden Timer wieder zurücksetzen und aktualisieren.
- Eine Optimierung des Spektrometers durchführen.
- Messung initiieren, sprich die Anzeige der PopUp View für den Messablauf.
- Verbindung zum Spektrometer trennen.

3.6.4.3 MeasurementSeries

In dieser Gruppe befindet sich die gesamte Logik, um Messungen in fest vorgegebenen Messprotokollen auszuführen. In der nachfolgenden Tabelle sind die drei Messprotokolle detailliert beschrieben.

Modus	Schritt 1	Schritt 2	Schritt 3
Raw	Target $[1 - n](0 - x)\{DC\}$		
Reflectance	White Reference (Ref) $[1](0 - x)$	Target $[1 - n](0 - x)$	
Radiance	White Reference (Rad) $[0 - n](0 - x)\{FO\}$	Target $[1 - n](0 - x)\{FO\}$	White Reference $[0 - n](0 - x)\{FO\}$

Erläuterungen zur Tabelle:

- $[n]$ = Anzahl Messungen die durchgeführt werden.
- (x) = Verzögerungen zwischen den Messungen.
- $\{DC\}$ = Dark Current Correction nicht durchführen
- $\{FO\}$ = Fore Optic einstellbar

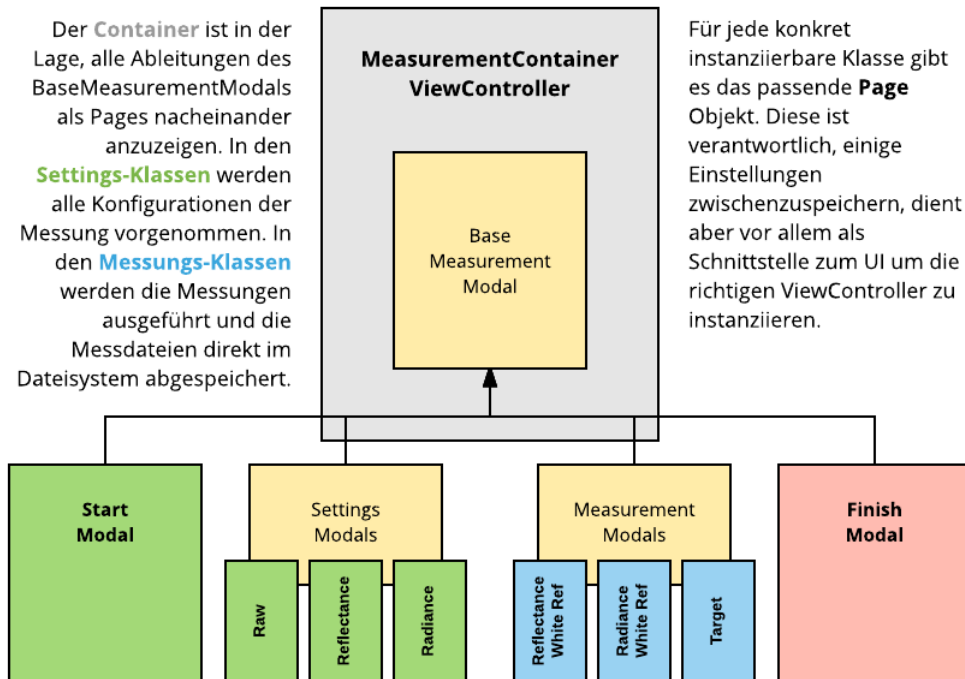


Abbildung 3.3: Feinarchitektur der Messreihen

Die Schritte 1-3 im Messprotokoll werden immer nacheinander durchgeführt. Ist beim Messprotokoll kein weiterer Schritt mehr vorhanden, führt dies automatisch zum Abschluss der Messung.

In der Abbildung 3.3 wird das ganze Zusammenspiel der Klassen dargestellt. Aus dieser Darstellung ist auch ersichtlich, dass für die White Reference Messung in den zwei Protokollen Reflectance und Radiance eine andere Klasse zur Ausführung kommt. Dies hat vor allem einen Grund, da die Views unterschiedliche Komponenten anbieten (ProgressBar), müssen sie getrennt verwaltet werden.

Um ein zusätzliches Messprotokoll umzusetzen, kann nun auf bestehende Komponenten zurückgegriffen werden oder für neue eine zusätzliche Klasse mit View erstellt werden. Zu beachten ist, dass bei einem zusätzlichen Messprotokoll auch eine neue Einstellungs-Klasse nötig wird.

Die Klassen Start, Settings und Finish werden hier nicht genauer erläutert, diese haben keinen grossen funktionellen Anteil und verwalten vor allem Einstellungen. Der genaue Ablauf in den MeasurementModals ist etwas komplizierter, folgt aber immer der Implementierung in der Basis Klasse. Die Grundidee ist, ein Liniendiagramm darzustellen, das fortlaufend immer aktuelle Messwerte beim Spektrometer anfordert. Entscheidet sich der Benutzer nun Messungen durchzuführen, werden die Messwerte zusätzlich abgefangen und entsprechend verarbeitet. Für die Darstellung werden die Messwerte entsprechend berechnet. In die effektiven Messdaten (Indico

Dateien) werden aber nur die gemessenen Raw-Daten geschrieben. Die wichtigsten Komponenten der Klasse `BaseMeasurementAquire` werden nachfolgend genauer beschrieben.

Die Funktion `startAquire()` wird direkt nach der Instanziierung der jeweiligen Klasse in der Methode `viewDidLoad()` aufgerufen. Mit diesem Aufruf beginnt das kontinuierliche Anfordern der Messdaten auf dem Spektrometer. Die Daten werden dann anhand der nachfolgend beschriebenen Implementierung verarbeitet und angezeigt.

```
func startMeasurement()
```

Diese Methode übernimmt alle Funktionalitäten sobald der Benutzer die Messwerte auch wirklich abspeichern möchte. In dieser Methode werden dann auch die entsprechenden UI Komponenten initialisiert, aktiviert oder deaktiviert.

```
func finishedMeasurement()
```

Sobald die eingestellte Anzahl an Messungen erreicht ist, wird diese Methode aufgerufen. Das weitere Abspeichern und Abzweigen von Messdaten wird automatisch gestoppt. Diese Methode kann überschrieben werden, sollten zum Messungsende weitere Schritte notwendig sein.

```
func handleRawSpectrum(currentSpectrum: FullRangeInterpolatedSpectrum)
```

Diese Methode **muss** von einer konkreten Implementierung überschrieben werden. Sie verarbeitet die Raw-Messdaten, dies bedeutet, die Daten werden entweder zwischengespeichert oder direkt in eine Datei geschrieben. Dies ist je nach Messprotokoll unterschiedlich.

```
func viewCalculationsOnCurrentSpectrum(currentSpectrum: FullRangeInte...)
```

Für die Diagrammanzeige müssen jeweils noch Berechnungen am Spektrum durchgeführt werden. Diese Methode **muss** ebenfalls zwingend überschrieben werden um die gewünschte Funktionalität zu garantieren.

```
func handleChartData(chartData: [Float])
```

Diese Methode **muss** ebenfalls zwingend überschrieben werden. Je nach Messprotokoll sollen Messdaten der vorherigen Messung noch bestehen bleiben oder gelöscht werden. Dieses Verhalten wird dadurch gesteuert. Die Methode verwaltet den `LineChartDataContainer`. In diesem werden die Chart-Daten entsprechend zwischengespeichert, die nach jedem Anfordern der Messdaten neu aktualisiert werden.

3.6.4.4 Measurements

Der `MeasurementTableViewController` und `MeasurementDetailViewController` implementieren die Funktionalität, die durchgeführten Messungen in einer Ordnerstruktur anzusehen. Die Table View ist von den erstellten Basiskomponenten `FileBrowser` abgeleitet und implementiert eine spezielle `MeasurementTableViewCell`. Diese bietet zusätzlich zur Navigation noch einen Export an.

Will der Benutzer einen Ordner oder eine Datei exportieren, wird ein `DiskFileAcitivityItem` erstellt und das Popup mit den Exportmöglichkeiten eingeblendet. Sobald der sein Exportziel ausgewählt hat, wird die Methode `activityViewController()` aufgerufen. Um einen gesamten

Ordner zu exportieren, wird zuerst eine temporäre ZIP Datei erstellt, die nach erfolgreichem Export direkt wieder gelöscht wird. Eine ASD Datei kann direkt exportiert werden.

3.6.4.5 Settings

Einstellungen, welche pro Applikation verfügbar sein müssen, werden in den sogenannten UserDefaults gespeichert. Dieser Speicher kann sämtliche serialisierbaren Objekte speichern.

3.6.5 Components

Die Erstellung von eigenen View Components, also Ableitungen von bestehenden Komponenten, hat in einem iOS Projekt den Vorteil, dass diese dann direkt im Storyboard verwendet und angezeigt werden können. So erhält man bereits bei der Entwicklung einen Eindruck über Anordnung, Farben und Grössenverhältnisse. Die Komponenten können mit eigenen Feldern ergänzt werden, die dann bequem im Storyboard eingestellt werden können.

Folgende Komponenten für das UI wurden selbst erstellt.

- SpectrometerTabBarItem: Das Icon kann direkt im Storyboard definiert werden.
- UIRoundBorderView: Einen ViewContainer mit einstellbaren abgerundeten Ecken.
- File- und Pathselects: Diese Komponenten rufen selbständig den Filebrowser auf und bieten ebenfalls Validierung an. Weiter gibt es eine Methode um mit dem selektierten File oder Path umzugehen.
- TextFields: Verschiedene Varianten wie Required, Optional, IP oder Port, die zusätzlich noch mit einem Icon versehen werden können.
- TitleSection: Wird in allen Modals im Titel Bereich verwendet.
- Buttons: RoundedColorButton, RadioButon oder LoadingButton die ebenfalls direkt im Storyboard abgeändert werden können.
- CustomChart: Anpassung der Chart View an die projektspezifischen Bedürfnisse.
- CustomProgressBar: Eine spezielle ProgressBar, die bei den Messungen zum Einsatz kommt.
- SettingsBox: Wird in den Messprotokoll-Einstellungen verwendet und stellt eine graphische Box dar.
- SelectFiberOptic: Ermöglicht alle verfügbaren FiberOptics einzublenden und anschliessend eine zu selektieren.

3.6.6 View Store

Der ViewStore dient wie der InstrumentStore zum Speichern weiter benötigter Variablen. Er ist ebenfalls als Singleton implementiert. Im Gegensatz zum InstrumentStore enthält der ViewStore aber nur Elemente, welche UI relevant sind. Darin wird beispielsweise gespeichert, ob gerade eine Aquire-Schleife läuft.

3.6.7 Service

3.6.7.1 Validation

Für die Validierung wurde von jedem benutzten Control eine Ableitung erstellt und das `BaseValidationControl` Protokoll implementiert. Dieses Protokoll enthält ein Feld `isValid` welches den Gültigkeitszustand des Objektes enthält.

Das Protokoll sieht wie folgt aus:

```
protocol BaseValidationControl {
    var isValid : Bool {get}
    func validate()
}
```

Ein Beispiel für die Implementierung des Protokolls ist das `PortTextField`:

```
class PortTextField : BaseTextField {
    //Init Code
    override var isValid: Bool {
        get {
            let port = Int(text!)
            return port != nil && port! >= 0 && port! <= 65535
        }
    }
}
```

In jedem ViewController muss nur noch der `ValidationManager` aufgerufen werden und die Haupt-View übergeben werden. Dieser `ValidationManager` prüft nun alle Sub-Views, welche das `BaseValidationControl` Protokoll implementieren. Dieser Aufruf sieht folgendermassen für die Validierung der gesamten View aus:

```
ValidationManager.sharedInstance.validateSubViews(view: self.view)
```

3.6.7.2 File Browser

Der File Browser ist eine Abwandlung von der Komponente **iOS FileBrowser in Swift**, welche unter folgendem Link zu finden ist: <https://github.com/marmelroy/FileBrowser>

Diese Komponente wurde auf die eigenen Bedürfnisse angepasst. Der File Browser, besteht jeweils aus einem Container sowie einem integrierten `TableViewController`. Zusammen bilden sie ein einfaches System, in welchem der Benutzer eine Datei oder einen Pfad selektieren kann.

Der File Browser kann einfach erweitert werden, indem ein neuer Container erstellt wird. Dieser Container muss von `FileBrowserContainerViewController` ableiten und kann diesen um neue Controls ergänzen. Um das Verhalten der inneren Tabelle zu ändern, kann eine Klasse erstellt werden, welche von `BaseFileBrowserTableViewController` ableitet. Dies wird im Projekt bereits genutzt, um den Container mit der Funktionalität der Ordnerwahl sowie der Ordnererstellung zu erweitern.

Folgend ein Beispiel, das für die Ordnerauswahl überschrieben wurde:

```
class DirectoryBrowserContainerViewController : FileBrowserContainerViewController {
    //Init

    @IBAction func ChooseDirectoryButtonClicked(_ sender: UIBarButtonItem) {
        didSelectFile!(DiskFile(url: selectedPath))
        dismiss(animated: true, completion: nil)
    }
}

class DirectoryBrowserTableViewController: FileBrowserTableViewController {
    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        let selectedFile = getFileForIndexPath(indexPath: indexPath)
        if(selectedFile.isDirectory)
        {
            // open folder
        }
        else
        {
            // deselect row
        }
    }
}
```

4 Testing

4.1 Unit Tests

Xcode und das iOS Framework stellen ein gutes UnitTest Framework zur Verfügung. Für das Spektrometer App wurden verschiedene Funktionen damit getestet. Die gesamten UniTests sind im Ordner `SpectrometerTests` zu finden.

4.1.1 IO Tests

Das gesamte Parsen, Lesen und Schreiben von Spektraldaten lässt sich optimal mit UnitTest testen. In der Klasse `SpectrometerIOTests` werden alle relevanten Funktionen dazu getestet. Nachfolgend werden die einzelnen Test Cases jeweils einzeln kurz beschreiben.

- `testParseSpectralData()`
Überprüft das korrekte parsen eines `FullRangeInterpolatedSpectrum` wie es vom Spektrometer gesendet wird.
- `testReadCalibrationFile()`
Überprüft das korrekte Einlesen einer INI-Datei.
- `testReadIndico7RawFile()`
Überprüft das korrekte Einlesen einer ASD-Messdatei im Raw Format
- `testReadIndico7ReflectanceFile()`
Überprüft das korrekte Einlesen einer ASD-Messdatei im Reflectance Format.
- `testReadIndico7RadianceFile()`
Überprüft das korrekte Einlesen einer ASD-Messdatei im Radiance Format.
- `testWriteRawData()`
Schreibt eine Messdatei im Raw Format und überprüft anschliessend ob diese korrekt wieder eingelesen werden kann.
- `testWriteReflectanceData()`
Schreibt eine Messdatei im Reflectance Format und überprüft anschliessend ob diese korrekt wieder eingelesen werden kann.
- `testWriteRadianceData()`
Schreibt eine Messdatei im Radiance Format und überprüft anschliessend ob diese korrekt wieder eingelesen werden kann.

4.2 Integration Tests

Alle Anforderungen wurden mit einem Testprotokoll getestet. Somit ist sichergestellt, dass alle Anforderungen den Vorgaben entsprechen und korrekt funktionieren. Das Testprotokoll ist im [Anhang](#) zu finden.

5 Projektorganisation

5.1 Vorgehen

Das Projekt wurde in den Grundsätzen an den RUP¹ Prozess angelegt. Als wichtigstes Merkmal wurden die Projektphasen Inception, Elaboration, Construction 1, Construction 2, Construction 3 und Transition definiert.

5.2 Risiken

Während des Projektes wurde festgestellt, dass die ASD Implementierung Eigenheiten besitzt, welche am Anfang unterschätzt wurden. Deshalb wurde eine Risikoanalyse vorgenommen und die Risiken des Projektes abgeschätzt. Die Risikobewertung wurde jeweils beim Übergang in eine neue Phase vorgenommen. Im folgenden Kapitel sind die Hauptrisiken aufgelistet.

5.2.1 ASD Dokumentation stimmt nicht mit Realität überein

Als Hauptrisiko mussten wir feststellen, dass die ASD Dokumentation nicht korrekt oder schwierig interpretierbar ist. Dieses Risiko ist eingetroffen. Als Massnahme wurde Code externer Entwickler und vom Kunden in Anspruch genommen, um die Daten besser interpretieren zu können.

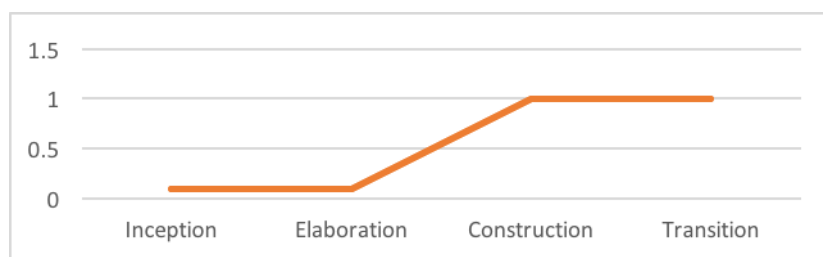


Abbildung 5.1: Risikoverlauf "ASD Dokumentation"

5.2.2 Zu wenig Zeit um alle Anforderungen zu erfüllen

Dieses Risiko wurde von Anfang an bewusst angegangen. So wurden bereits während der Anforderungserhebung eine Priorisierung der einzelnen Punkte vorgenommen. Dieser Prozess erfolgte in Zusammenarbeit mit dem Kunden. Dadurch konnte dieses Risiko minimiert werden.

¹Rational Unified Process: https://de.wikipedia.org/wiki/Rational_Unified_Process

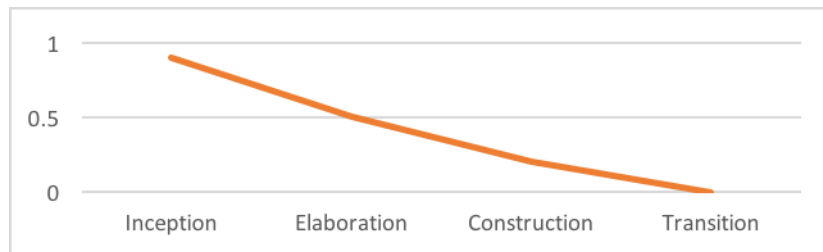


Abbildung 5.2: Risikoverlauf "Zeit"

5.2.3 Daten vom iOS Gerät importieren und exportieren

Dieses Risiko bestand, da im Gegensatz zu vielen etablierten Apps einige Dateien zwingend für das Funktionieren der Applikation bereits zu Beginn bereitgestellt werden müssen. Die Sorge war, keine einfache Methode anbieten zu können, um einfach und intuitiv Daten auf das iPad zu kopieren. Dieses Risiko konnte ausgeräumt werden, indem Nachforschungen zu bekannten Apps, welche diese Problemstellung gelöst hatten, intensiviert wurden. Es wurde festgestellt, das iOS bereits eine eigene elegante Lösung für diesen Prozess bietet, welcher nun auch verwendet wird. Das Importieren wird in Punkt 3.6 beschrieben.

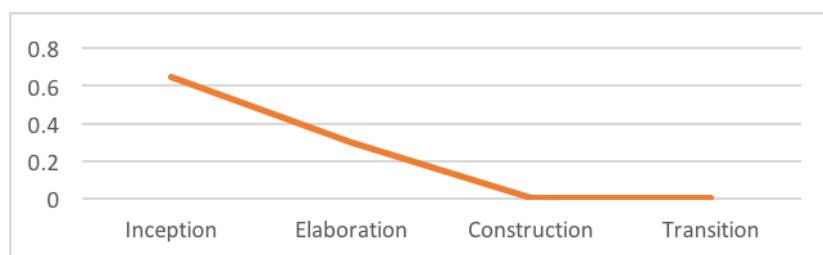


Abbildung 5.3: Risikoverlauf "Import und Export"

5.3 Meilensteine

Die Meilensteine wurden ebenfalls an den RUP Prozess angelegt und sind so meist bei den Übergängen in die nächste Phase definiert. In der nachfolgenden Tabelle sind die definierten Meilensteine des Projektes nach Datum geordnet:

Nr.	Datum	Beschreibung
001	23.10.2016	Einlesen und Spektrometerübergabe Die Spektrometer konnten von den Studierenden in Empfang genommen werden.
002	30.11.2016	Anforderungen Das Pflichtenheft wurde erstellt und vom Kunden akzeptiert.
003	30.11.2016	Proof of Concept Es liegt ein funktionierender Proof of Concept vor, der die Verbindung zum Spektrometer herstellen und Antworten empfangen kann.
004	21.12.2016	Prototyp 1 Ein funktionierender Prototyp mit allen Anforderungen der Priorität 1 ist für den Kunden im TestFlight freigegeben.
005	25.01.2017	Prototyp 2 Ein funktionierender Prototyp mit allen Anforderungen der Priorität 2 ist für den Kunden im TestFlight freigegeben.
006	01.03.2017	Version 1.0 Eine funktionierende Version der App mit allen Anforderungen der Priorität 3 ist für den Kunden im TestFlight freigegeben.
007	16.03.2017	Projektabschluss Die finale Version mit allen Fehlerverbesserungen aus Version 1.0 ist im TestFlight für den Kunden freigegeben.

5.4 Zeitplanung

Die Zeitplanung wurde mithilfe der RUP Phasen durchgeführt. Damit konnte die Dauer der einzelnen Phasen abgeschätzt und mit den Meilensteinen abgestimmt werden. Es musste noch auf einige Abwesenheiten des Kunden geachtet werden. Die Meetings für die Prototyp Präsentation mussten somit etwas vor- oder nach den jeweiligen Releasedaten der Prototypen angesetzt werden. Der Zeitplan befindet sich in detaillierter Form auch im Anhang.

5.5 Anforderungen

Die Anforderungen wurden zu Beginn bei einem Startmeeting mit dem Kunden besprochen. Weiter konnten viele Anforderungen detailliert in der bestehenden Software ausgemacht werden. Die Anforderungen wurden in einem Pflichtenheft zentral erfasst und priorisiert. Die Prioritäten

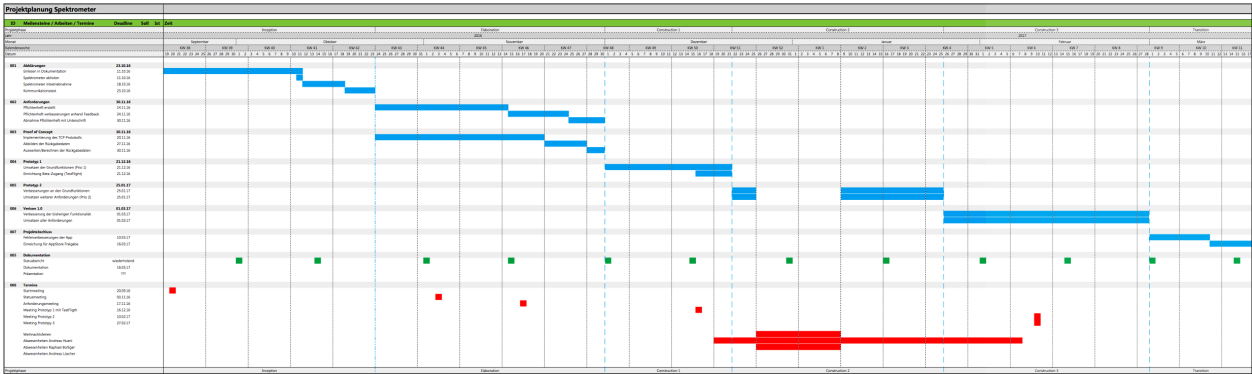


Abbildung 5.4: Zeitplanung

markieren zudem, in welchem Prototyp eine Anforderung umgesetzt wurde. Anforderungen mit der Priorität 4 wurden nicht zwingend umgesetzt, diese haben keinen Einfluss auf die Funktionalität der neuen Applikation. Detaillierte Informationen zu den Anforderungen sind im Anhang zu entnehmen.

5.6 Change Management

Die Anforderungen haben sich bis kurz vor Projektende nie geändert. Nach dem Einreichen der Version 1.0 kam die Anforderung dazu, den Dark Current mit einer Konfigurationsdatei zu hinterlegen sowie zu Berechnen. Da diese so kurzfristig war, konnte sie nicht mehr innerhalb der Projektzeit umgesetzt werden.

Auf ein Change Management wurde bewusst verzichtet, da sich nach Projektstart abzeichnete, dass die Anforderungen genug detailliert und vollständig erfasst wurden.

5.7 Arbeitspakete

Da die Anforderungen sehr detailliert unterteilt wurden, dienten sie zugleich als Arbeitspakete. Eine Anforderung konnte so meist von einer Person implementiert werden.

5.8 Soll- und Istvergleich

Beim Projektende wurde ersichtlich, dass die Zeiten zum Teil etwas falsch eingeschätzt wurden. In der Planungsphase wurde viel Zeit eingerechnet diese konnte dort aber kaum genutzt werden, da zu viele Unklarheiten offen waren. Am Schluss beim finalen Prototypen konnte der Kunde noch wichtige Feedbacks liefern womit sich die benötigte Zeit dort etwas erhöhte. Weiter wurde die Projektdauer kurzfristig um eine Woche verlängert womit die Gesamtzeit ebenfalls etwas höher ausfällt.

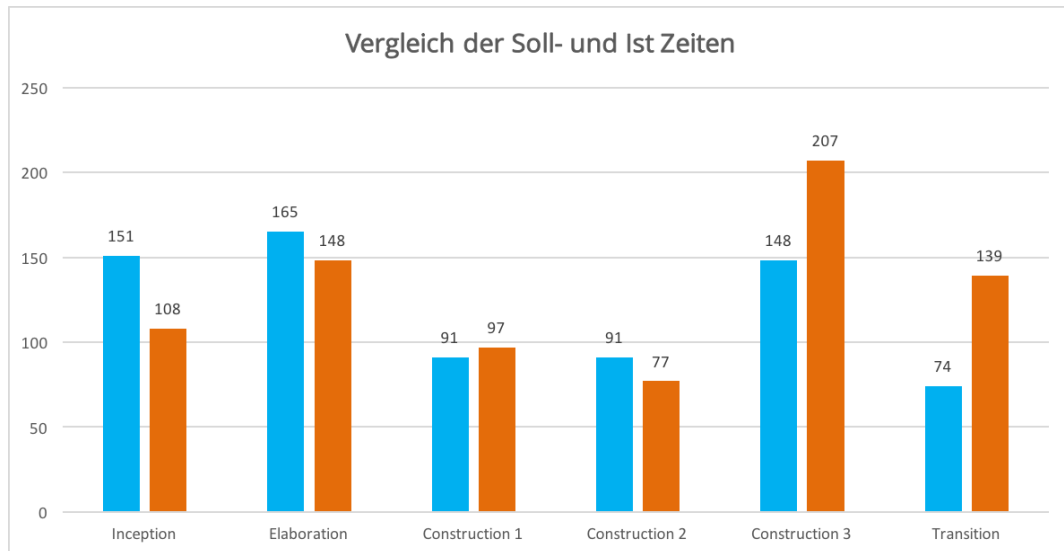


Abbildung 5.5: Soll- und Istvergleich

6 Fazit

6.1 Zusammenfassung

Das Projektteam hatte anfangs Schwierigkeiten, mithilfe der ASD-Dokumentation eine Verbindung zum Gerät aufzubauen und korrekte Daten zu empfangen. Nach dieser Hürde konnten die Anforderungen gut umgesetzt werden. Die Aufteilung in die einzelnen Layer, machte die Entwicklung einfacher und half, die Übersicht über den grösser werdenden Projektumfang zu bewahren.

6.2 Potentielle Erweiterungen

6.2.1 Neues ASD Gerät

Die wohl wahrscheinlichste Erweiterung in Zukunft ist das Erweitern der Applikation, um die Kompatibilität mit neueren ASD Geräten herzustellen. Eine solche Erweiterung wurde bereits während der Entwicklungszeit in Betracht gezogen. Der bestehende Code kann erweitert werden, ohne dessen Funktionalität anpassen zu müssen.

6.2.2 Absolut Reflectance

Diese Erweiterung wurde als Priorität 4 aufgenommen, konnte allerdings im Projektrahmen nicht mehr umgesetzt werden. Jedoch wird bereits das Auswählen und Speichern eines Absolut-Reflectance-Files in der Applikation angeboten. Dieses muss nur noch in die Berechnung der Anzeige einfließen.

6.2.3 GPS Daten und Fotos zu Messungen hinzufügen

Diese Erweiterung wurde ebenfalls als Requirement mit geringer Priorität aufgenommen und ist nicht umgesetzt worden. Das Indico File Format bietet für die GPS Daten bereits einen entsprechenden Abschnitt. Dieser ist in der FileWriter Klasse auch beschrieben und wird momentan übersprungen. Für die Speicherung von Bildern, müsste das Indico Protokoll erweitert werden. Auf die GPS Daten und die Kamera des Gerätes kann einfach über die von Apple vorgesehenen Schnittstellen zugegriffen werden.

6.3 Erkenntnisse

Trotz der vorhandenen Dokumentation, war das Projekt nicht einfach zu realisieren. Insbesondere die Verbindungsherstellung und die Eigenheiten des ASD Geräts, gestalteten den Einstieg in das Projekt schwieriger als erwartet. Auch die Abläufe der Messungen fühlen sich anfangs teilweise widersprüchlich an. Ist die Abfolge allerdings geklärt und die Verbindung korrekt umgesetzt, ist die Implementierung der Anforderungen relativ zügig machbar.

Auch sprachbedingt mussten einige Anpassungen vorgenommen werden, welche bei anderen Sprachen nicht benötigt worden wären. So können in Swift keine Fatal Errors, beispielsweise bei einem Array out of Index Fehler, abgefangen werden. Dies macht Sinn, wenn ein Produkt aus einer Hand entwickelt wird. Bei unserer Abhängigkeit zur Rückgabe des Spektrometers, kann es passieren, dass die Grösse oder Typen der Rückgabe nicht stimmen. Diese Fehler müssen speziell abgefangen werden und können nicht einfach generell behandelt werden. Weiter können in Swift keine abstrakten Klassen definiert werden. Somit können externe Entwickler versehentlich vergessen Methoden zu überschreiben. Dies wurde gelöst, indem in den Basismethoden welche abgeleitet werden müssen ein Fatal Error geworfen wird.

7 Ehrlichkeitserklärung

Hiermit bestätigen die unterzeichnenden Autoren, dass alle nicht klar gekennzeichneten Stellen von ihnen selbst erarbeitet und verfasst wurden.

Brugg, 23 März 2017

Raphael Bolliger

Unterschrift

Andreas Lüscher

Unterschrift

8 Abbildungsverzeichnis

2.1	Inhalt der Messdateien	3
2.2	Berechnungen für Raw, Reflectance und Radiance	4
3.1	Systemgrenzen	7
3.2	Layeraufteilung der Grobarchitektur	9
3.3	Feinarchitektur der Messreihen	16
5.1	Risikoverlauf "ASD Dokumentation"	22
5.2	Risikoverlauf "Zeit"	23
5.3	Risikoverlauf "Import und Export"	23
5.4	Zeitplanung	25
5.5	Soll- und Istvergleich	26

9 Glossar

Auto Layout Anhand von Constraints wird die Grösse und Position von Elementen auf der View berechnet: [Auto Layout Guide](#).

Dark Current Als Dark Current bezeichnet man ein Spektrum, welches aufgenommen wurde ohne Lichteinfall. Der Darkcurrent bezieht sich nur auf das visuelle Spektrum (vinir).

DN DC_CORR Digital Numbers Dark Corrected, sprich das Resultat der Raw Messung nach der Dark Current Correction.

DN WR_DC_CORR Digital Numbers der White Reference Dark Corrected, sprich das Resultat der White Reference Messung nach der Dark Current Correction.

Faser Ist ein Kabel mit dem das Licht für die Messung im Spektrometer eingefangen wird.

IDE Integrierte Entwicklungsumgebung:
https://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung.

Spektrometer Ist ein Gerät mit welchem in der Natur Spektraldaten gemessen werden können.

SWIR1 Shortwave Infrared, kurzwellige Infrarotstrahlung. Eins steht für den ersten Sensor..

SWIR2 Shortwave Infrared, kurzwellige Infrarotstrahlung. Zwei steht für den zweiten Sensor..

TestFlight Service von Apple um unkompliziert Betaversionen auszuliefern:
<https://developer.apple.com/testflight/>.

VNIR Visible and near-infrared Bereich des Spektrums.

White Reference Eine Messung über einer bestimmten Weissreferenz-Platte.

10 Literaturverzeichnis

- [1] Apple Inc. Core data programming guide.
<https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData>.
[Abgerufen am: 22.11.2016].
- [2] Pietro Rea. Getting started with core data tutorial.
<https://www.raywenderlich.com/145809/getting-started-core-data-tutorial>.
[Abgerufen am: 22.11.2016].
- [3] Nate Cook. Ibinspectable / ibdesignable.
<http://nshipster.com/ibinspectable-ibdesignable>.
[Abgerufen am: 26.01.2017].
- [4] Gabriel Theodoropoulos. Grand central dispatch (gcd) and dispatch queues in swift 3.
<http://www.appcoda.com/grand-central-dispatch>.
[Abgerufen am: 23.01.2017].
- [5] Andrew Schreiber. Array extensions in swift 3.
<http://stackoverflow.com/a/33557647>.
[Abgerufen am: 06.03.2017].
- [6] Apple Inc. Auto layout guide.
<https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/AutolayoutPG>.
[Abgerufen am: 12.01.2017].
- [7] Andreas Hueni. Specchio github repository.
<https://github.com/ahueni/SPECCHIO>.
[Abgerufen am: 20.02.2017].

11 Anhang

In der ausgedruckten Version wird, aufgrund des Umfangs, auf den Inhalt des Anhangs verzichtet.
Der komplette Anhang ist in der digitalen Version vorhanden.

11.1 Zeitplan

11.2 Risikoanalyse

11.3 Testprotokolle

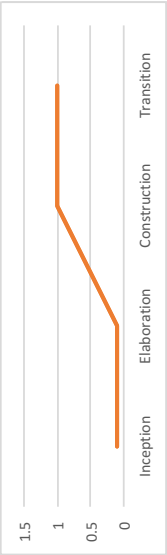
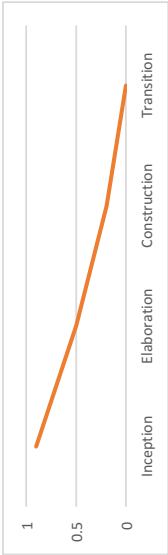
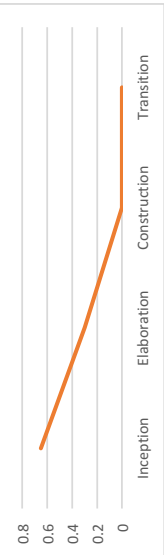
11.4 ASD

11.4.1 Indico File Format

11.4.2 ASD Developers Guide

Project planning: Spectrometer									
ID	Task/Module / Subtask / Feature	Duration	Start	End	2024	2025	2026	2027	2028
Phase 1: Planning & Design (2024)									
P1.1: Project Kick-off									
P1.2: Requirements Gathering									
P1.3: System Architecture Design									
P1.4: Detailed Design									
P1.5: Review & Approval									
Phase 2: Development & Testing (2024-2025)									
D2.1: Software Development									
D2.2: Hardware Development									
D2.3: Integration Testing									
D2.4: User Acceptance Testing									
D2.5: Performance Testing									
D2.6: Security Testing									
D2.7: Reliability Testing									
D2.8: Compatibility Testing									
D2.9: Documentation									
Phase 3: Deployment & Support (2025-2028)									
D3.1: Deployment Planning									
D3.2: Deployment Execution									
D3.3: Post-Deployment Support									
D3.4: System Maintenance									
D3.5: System Upgrades									
D3.6: System Decommissioning									

Risikoanalyse

Name	Phase	Wahrscheinlichkeit	Massnahmen	Verlauf
ASD Dokumentation stimmt nicht mit Realität überein	Inception	10%	Keine	
	Elaboration	10%	Keine	
	Construction	100%	Externer Code miteinbezogen.	
	Transition	100%	Externer Code miteinbezogen.	
Zu wenig Zeit um alle Anforderungen zu erfüllen	Inception	90%	Priorisierung der Anforderungen	
	Elaboration	50%		
	Construction	20%		
	Transition	0%		
Daten vom iOS Gerät importieren und exportieren	Inception	65%	Andere Applikationen mit der selben Problemstellung angesehen.	
	Elaboration	30%	Prototyp für das Importierens erstellt	
	Construction	0%		
	Transition	0%		

Test Protokoll

Einleitung

In diesem Dokument werden alle manuellen Testfälle aufgelistet, welche für das Spektrometer App durchgeführt wurden.

Testbedingungen

Die Tests wurden mit dem FieldSpec 3 Spektrometer durchgeführt. Die Applikation lief jeweils auf dem iOS Simulator oder auf einem iPad Air.

Voraussetzungen

Das Testgerät (Simulator oder iOS Gerät) muss eine Verbindung zum ASD Spektrometer aufgebaut haben.

In den Einstellungen muss eine gültige IP und Subnetzmaske gesetzt worden sein.

Es muss vorgängig mindestens ein Base, ein Lamp und ein Foreoptic File importiert worden sein.

Was wird getestet

Alle Anforderungen, welche im Pflichtenheft definiert sind.

Was wird nicht getestet

Testfälle welche sich auf das Testen des Betriebssystems beziehen, werden nicht getestet.

Beispielsweise das Importieren von Files.

Ebenfalls wird kein externer Code getestet. Bei der Auswahl von externen Libraries wurde darauf geachtet, dass nur mehrfach eingesetzte Frameworks verwendet werden.

Vorlage des Testprotokolls

Mit dieser Vorlage können weitere Testfälle abgebildet werden.

Vorlage			Spektrometer App		
Test ID			Durchgeführt von		
Test Titel			Datum		
Anforderung			Testgerät		
Beschreibung					
Bedingungen					
Abhängigkeiten					
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1					<input checked="" type="checkbox"/> <input type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0001		Durchgeführt von	Andreas Lüscher	
Test Titel	Verbindung initialisieren		Datum	08.03.2017	
Anforderung	1.0.001		Testgerät	Fieldspec 4 und Simulator	
Beschreibung	Herstellen einer Verbindung mit dem Gerät				
Bedingungen	Mindestens eine Verbindung muss konfiguriert sein.				
Abhängigkeiten					
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Verbindung auswählen und Verbinden anklicken		Verbindung wird erfolgreich hergestellt. Weiterleitung auf Verbindungsseite	Verbindung wird hergestellt. Weiterleitung erfolgt.	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0002		Durchgeführt von	Andreas Lüscher	
Test Titel	Verbindungseinstellungen erstellen		Datum	08.03.2017	
Anforderung	1.0.002		Testgerät	Fieldspec 4 und Simulator	
Beschreibung	Im App kann ein neues Spektrometer mit Verbindungseinstellungen erstellt werden.				
Bedingungen					
Abhängigkeiten					
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	„Add Spectrometer“ antippen		Pop-Up erscheint um eine neue Verbindungseinstellung zu erstellen	Pop-Up erscheint korrekt	<input checked="" type="checkbox"/>
2	Namensfeld ausfüllen	Fieldspec 4			<input checked="" type="checkbox"/>
3	IP Adresse ausfüllen	10.1.1.77			<input checked="" type="checkbox"/>
4	Port ausfüllen	8080			<input checked="" type="checkbox"/>
5	Select Base File antippen und Base File auswählen	Base File	Filebrowser erscheint	Filebrowser erscheint	<input checked="" type="checkbox"/>
6	Select Lamp File antippen und Lamp File auswählen	Lamp File	Filebrowser erscheint	Filebrowser erscheint	<input checked="" type="checkbox"/>
7	Add Foreoptic antippen und Foreoptic auswählen	Foreoptic File	Filebrowser erscheint	Filebrowser erscheint	<input checked="" type="checkbox"/>
8	Schritt 7 für alle Foreoptic Files wiederholen	Foreoptic File	Filebrowser erscheint und eine zusätzliche Foreoptic wird angezeigt nach der Auswahl	Filebrowser erscheint und Foreoptic wird hinzugefügt	<input checked="" type="checkbox"/>
9	Speichern		Verbindung sollte gespeichert sein und in der Tabelle angezeigt werden	Verbindung wurde erfolgreich gespeichert	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0003		Durchgeführt von	Andreas	
Test Titel	Verbindungseinstellungen löschen		Datum	08.03.2017	
Anforderung	1.0.002		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Verbindungseinstellungen löschen				
Bedingungen	Mindestens eine Verbindung muss konfiguriert sein				
Abhängigkeiten					
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Auf der Verbindungsübersicht mit einem Swipe nach links das Kontext Menü einer Verbindung aufrufen		Kontextmenü erscheint	Kontextmenü erscheint	<input checked="" type="checkbox"/>
2	Auf Löschen tippen		Verbindung wird gelöscht und erscheint nicht mehr in der Tabelle	Verbindung wird erfolgreich gelsöcht	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0004		Durchgeführt von	Andreas	
Test Titel	Verbindungseinstellungen bearbeiten		Datum	08.03.2017	
Anforderung	1.0.002		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Verbindungseinstellungen bearbeiten				
Bedingungen					
Abhängigkeiten					
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Spektrometer auswählen und mit einem Swipe nach link das Kontextmenü öffnen		Kontextmenü erscheint	Kontextmenü erscheint	<input checked="" type="checkbox"/>
2	Auf Bearbeiten tippen		Pop-Up erscheint	Pop-Up erscheint	<input checked="" type="checkbox"/>
3	IP ändern	IP: 10.1.1.78			<input checked="" type="checkbox"/>
4	Port ändern	Port: 8082			<input checked="" type="checkbox"/>
5	Bestehendes Base File antippen	Base File	Filebrowser erscheint	Filebrowser erscheint	<input checked="" type="checkbox"/>
6	Bestehendes Lamp File antippen	Lamp File	Filebrowser erscheint	Filebrowser erscheint	<input checked="" type="checkbox"/>
7	Add Foreoptic antippen und zusätzliche Foreoptic auswählen	Foreoptic File	Filebrowser erscheint und eine zusätzliche Foreoptic wird angezeigt nach der Auswahl	Filebrowser erscheint und Foreoptic wird hinzugefügt	<input checked="" type="checkbox"/>
8	Schritt 7 für alle Foreoptic Files wiederholen	Foreoptic File	Filebrowser erscheint und eine zusätzliche Foreoptic wird angezeigt nach der Auswahl	Filebrowser erscheint und Foreoptic wird hinzugefügt	<input checked="" type="checkbox"/>
9	Speichern		Verbindung wird mit neuen Daten gespeichert.	Verbindung wurde erfolgreich geändert	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0006		Durchgeführt von	Andreas	
Test Titel	Einstellen der Foreoptic		Datum	08.03.2017	
Anforderung	2.0.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Im App muss die Foreoptic gewechselt werden können				
Bedingungen	Mindestens 2 Foreoptic Files müssen importiert und der Verbindung hinzugefügt worden sein				
Abhängigkeiten	Test ID 2 durchgeführt mit mindestens 2 Foreoptic Files, danach Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Auf dem Startscreen auf Foreoptic tippen		Alle Foreoptic Files welche eingelesen sind, werden mit Namen angezeigt	Alle Foreoptic Files werden mit korrektem Namen angezeigt	<input checked="" type="checkbox"/>
2	Foreoptic auswählen		Die Foreoptic wird gesetzt	Die Foreoptic wird korrekt gesetzt	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0007		Durchgeführt von	Andreas	
Test Titel	Einstellen der Number of Samples		Datum	08.03.2017	
Anforderung	2.0.002		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Die Anzahl Samples muss eingestellt werden können. Jeweils für eine aquire eine DC und eine WR Messung.				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Auf den Einstellungstab tippen		Einstellungen werden geöffnet	Einstellungen werden geöffnet	<input checked="" type="checkbox"/>
2	Auf Instrument Configuration tippen		Instrument Configuration wird in der DetailView angezeigt.	Instrument Configuration erscheint.	<input checked="" type="checkbox"/>
3	Den Aquire Sample Slider verwenden	Aquire Sample auf 20 setzten			<input checked="" type="checkbox"/>
4	Den DarkCurrent Sample Slider verwenden	DC Sample auf 20 setzten			<input checked="" type="checkbox"/>
5	Den WhiteReference Sample Slider verwenden	WR Sample auf 20 setzten			<input checked="" type="checkbox"/>
6	Einstellungen verlassen -> Auf Spectrometer Tab tippen.				
7	Schritt 2- 3 wiederholen		Alle Werte werden korrekt geladen.	Die gespeicherten Werte werden korrekt geladen	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0008		Durchgeführt von	Andreas	
Test Titel	Einstellen der Visible Parameter		Datum	08.03.2017	
Anforderung	2.0.003		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Der Shutter muss manuell geschlossen und die Integrationszeit muss eingestellt werden können.				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Auf den Einstellungstab tippen		Einstellungen werden geöffnet	Einstellungen werden geöffnet	<input checked="" type="checkbox"/>
2	Auf Instrument Control tippen		Instrument Control wird in der DetailView angezeigt.	Instrument Control erscheint.	
3	Shutter ausschalten	Shutter disablen			<input checked="" type="checkbox"/>
4	Integrationszeit ändern	Integrationszeit auf 17 ms setzen			<input checked="" type="checkbox"/>
5	Einstellungen verlassen -> Auf Spectrometer Tab tippen.				
6	Messungslöop starten		Alle Messungen empfangen keine Vinir Daten, da der Shutter geschlossen ist.		
7	Schritt 2- 3 wiederholen		Alle Werte werden korrekt geladen.	Die gespeicherten Werte werden korrekt geladen	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0009		Durchgeführt von	Andreas	
Test Titel	Einstellen der „Nir, Swir1 und Swir2“ Parameter		Datum	08.03.2017	
Anforderung	2.0.004		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Gain und Offset für Nir, Swir1 und Swir 2 muss manuell gespeichert werden können.				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
2	Auf den Einstellungs-Tab tippen		Einstellungen werden geöffnet	Einstellungen werden geöffnet	<input checked="" type="checkbox"/>
3	Auf Instrument Control tippen		Instrument Control wird in der Detail-View angezeigt.	Instrument Control erscheint.	<input checked="" type="checkbox"/>
4	Integration-Time setzen	Wert:			<input checked="" type="checkbox"/>
5	Swir 1 Gain setzen	Wert: 500			<input checked="" type="checkbox"/>
6	Swir 1 Offset setzen	Wert: 100			<input checked="" type="checkbox"/>
7	Swir 2 Gain setzen	Wert: 500			<input checked="" type="checkbox"/>
8	Swir 2 Offset setzen	Wert: 100			<input checked="" type="checkbox"/>
9	Update antippen		Die Werte werden auf dem Gerät gespeichert.	Die gespeicherten Werte sind auf dem Gerät korrekt hinterlegt.	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0010		Durchgeführt von	Andreas	
Test Titel	Einstellen des Scan-Type		Datum	08.03.2017	
Anforderung	2.0.005		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen	Diese Anforderung wurde noch nicht umgesetzt (Priorität 4). Wird erst nach der Umsetzung getestet.				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Auf den Einstellungs-Tab tippen		Einstellungen werden geöffnet	Einstellungen werden geöffnet	
2	Auf Instrument Control tippen		Instrument Control wird in der Detail-View angezeigt.	Instrument Control erscheint.	
3	Scan Type setzen	Scan Type A only wählen			
4	Messung auslösen		Der ScanType wird mitgesendet	Im Acquire Befehl, ist der gewählte Scan Type gesetzt	

Testprotokoll			Spektrometer App		
Test ID		0011	Durchgeführt von		Andreas
Test Titel		Einstellen der „Absolute Reflectance“	Datum		08.03.2017
Anforderung		2.0.005	Testgerät		Fieldspec 3 / Simulator
Beschreibung					
Bedingungen		Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.			
Abhängigkeiten		Test 1 durchführen			
Bemerkungen		Diese Anforderung wurde noch nicht umgesetzt (Priorität 4). Wird erst nach der Umsetzung getestet.			
Nr.	Schritte		Testdaten		Erwartetes Resultat
					Resultat
					Erfüllt

Testprotokoll			Spektrometer App		
Test ID	0012		Durchgeführt von	Andreas	
Test Titel	„Dark Current“ auslösen		Datum	08.03.2017	
Anforderung	3.0.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Auf dem Spektrometer soll ein Dark Current ausgelöst werden können				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Dark Current antippen		Der Dark Current wird ausgelöst und bei weiteren Messungen abgezogen.	Dark Current wurde ausgelöst	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0013		Durchgeführt von	Andreas	
Test Titel	„White Reference“ auslösen		Datum	08.03.2017	
Anforderung	3.0.002		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Auf dem Spektrometer soll eine White Reference Messung ausgelöst werden können				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	White Reference antippen		Es wird eine White Reference Messung durchgeführt. Der WR Timer wird zurückgesetzt und beginnt wieder von 0 Sekunden an zu ticken.	Die WR Messung wurde korrekt durchgeführt	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0014		Durchgeführt von	Andreas	
Test Titel	Spectromearument auslösen		Datum	08.03.2017	
Anforderung	3.0.003		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Es wird kontinuierlich eine Messung ausgelöst				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Start Acquire tippen		Es wird kontinuierlich eine Messung ausgelöst und im Diagramm angezeigt	Die Messungen werden korrekt nacheinander ausgelöst.	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0015		Durchgeführt von	Andreas	
Test Titel	Optimize Instrument auslösen		Datum	22.03.2017	
Anforderung	3.0.004		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Auf der Acquire View wird ein Optimize ausgelöst.				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Optimize antippen		Es wird ein Optimize und ein Dark Current ausgelöst. Die WR Daten werden gelöscht. Reflectance ist nicht mehr verfügbar	Optimize wird ausgelöst und Gains sowie die Integration Time wird angezeigt.	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0016		Durchgeführt von	Andreas	
Test Titel	Abbrechen der Messungen		Datum	08.03.2017	
Anforderung	3.0.005		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Das kontinuierliche Messen wird unterbrochen.				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 14				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Stopp antippen		Die Messschleife wird abgebrochen.	Die Messschleife stoppt.	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0017		Durchgeführt von	Andreas	
Test Titel	Anlegen einer Messung		Datum	08.03.2017	
Anforderung	4.0.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Anlegen einer Messung im Raw-Format und einstellen aller Parameter.				
Bedingungen	Verbindung mit dem Spektrometer wurde erfolgreich hergestellt.				
Abhängigkeiten	Test 1 durchführen, Test 12 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messung starten antippen		Das Messfenster erscheint	Das Messfenster wird erfolgreich aufgerufen.	<input checked="" type="checkbox"/>
2	Name setzen	Name: „Neue Messung“			<input checked="" type="checkbox"/>
3	Kommentar setzen	Kommentar: „Test“			<input checked="" type="checkbox"/>
4	Ordner auswählen	Neuen Ordner „Messungen“ erstellen.	File Browser soll geöffnet werden	File Browser wird geöffnet	<input checked="" type="checkbox"/>
5	Modus wählen	Modus „Raw“ auswählen			<input checked="" type="checkbox"/>
6	Weiter antippen		Es wird auf die Rawsettings Seite gewechselt	Raw Settings werden angezeigt	<input checked="" type="checkbox"/>
7	Anzahl Messungen wählen	5 Messungen			<input checked="" type="checkbox"/>
8	Intervall wählen	2 Sekunden			<input checked="" type="checkbox"/>
9	Starten antippen		Messseite wird geöffnet.	Messseite wird geöffnet	<input checked="" type="checkbox"/>
10	Messung starten antippen		Messungen werden durchgeführt	Messungen werden durchgeführt	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0018		Durchgeführt von	Andreas	
Test Titel	Abspeichern der Daten im Indigo File		Datum	08.03.2017	
Anforderung	4.0.002		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Abgespeicherte Daten können im Browser wieder angezeigt werden.				
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Mit dem Anzeigen wird gezeigt, dass die Daten im Indigo Format gespeichert wurden.				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Messungen ansehen		Alle Messungen werden bei einem Tap erfolgreich gelesen und angezeigt	Alle Messungen werden bei einem Tap erfolgreich gelesen und angezeigt	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0019		Durchgeführt von	Andreas	
Test Titel	Export der Daten		Datum	08.03.2017	
Anforderung	4.1.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Einzelnes File exportieren				
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Ein File wird direkt exportiert. Mehrere Files werden als Zip Ordner exportiert.				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Teilen Button eines Files anklicken		Ein Pop Up erscheint um die Daten zu exportieren	Pop-Up erscheint	<input checked="" type="checkbox"/>
4	Andere App auswählen	Mail App auswählen	Die Mail App wird geöffnet und das File wird exportiert	Das File wird erfolgreich exportiert	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0020		Durchgeführt von	Andreas	
Test Titel	Export der Daten		Datum	08.03.2017	
Anforderung	4.1.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Ordner exportieren				
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Ein File wird direkt exportiert. Mehrere Files werden als Zip Ordner exportiert.				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Teilen Button eines Ordners anklicken		Ein Pop Up erscheint um die Daten zu exportieren	Pop-Up erscheint	<input checked="" type="checkbox"/>
4	Andere App auswählen	Mail App auswählen	Die Mail App wird geöffnet und die Files werden als Zip komprimierter Ordner übergeben	Der Zip Ordner wurde erfolgreich exportiert	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0021		Durchgeführt von	Andreas	
Test Titel	Verwalten der konfigurierten Messdaten		Datum	08.03.2017	
Anforderung	4.2.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung	Messdaten sollen gelöscht werden können				
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Edit antippen		Bearbeiten Modus wird aktiviert	Bearbeiten Modus wird aktiviert	<input checked="" type="checkbox"/>
4	Löschbutton eines Files antippen	Erste Messdatei	Die Messung wird gelöscht und aus der Tabelle entfernt.	Die Messung wird gelöscht und aus der Tabelle entfernt.	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0022		Durchgeführt von	Andreas	
Test Titel	Ergänzung der Messungen mit GPS Daten		Datum	08.03.2017	
Anforderung	5.0.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Prio 4: Nicht implementiert				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	
3	Detail aufrufen	Erste Messung	Details der Messung werden angezeigt	Details der Messung werden angezeigt	
4	GPS anzeigen antippen		Karte mit Messkoordinaten wird angezeigt	Karte mit Messung wird geöffnet	

Testprotokoll			Spektrometer App		
Test ID	0023		Durchgeführt von	Andreas	
Test Titel	Ergänzung der Messungen mit Fotos		Datum	08.03.2017	
Anforderung	5.1.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Prio 4: Nicht implementiert				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	
3	Detail aufrufen	Erste Messung	Details der Messung werden angezeigt	Details der Messung werden angezeigt	
4	Foto anzeigen antippen		Foto der Messumgebung wird angezeigt	Foto der Messumgebung wird angezeigt	

Testprotokoll			Spektrometer App		
Test ID	0024		Durchgeführt von	Andreas	
Test Titel	Darstellung der „raw DN“ Werte		Datum	08.03.2017	
Anforderung	6.0.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Erste Messung antippen	Erste Messung	Modus ist Raw	Modus ist auf Raw eingestellt	<input checked="" type="checkbox"/>
4			X-Achse verläuft von 350 bis 2500	X: 305 – 2500	<input checked="" type="checkbox"/>
5			Y-Achse verläuft von 0 bis 65000	Y: 0 – 65000	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0025		Durchgeführt von	Andreas	
Test Titel	Darstellung der „Radiance“ Werte		Datum	08.03.2017	
Anforderung	6.0.002		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Falscher Wert im Pflichtenheft hinterlegt, die Y-Achse sollte von 0 bis und mit 1 gehen.				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Erste Messung antippen	Erste Messung	Modus ist Raw	Modus ist auf Raw eingestellt	<input checked="" type="checkbox"/>
4	Modus Rad antippen	Erste Messung	Modus ist Radiance	Modus ist auf Rad eingestellt	<input checked="" type="checkbox"/>
5			X-Achse verläuft von 350 bis 2500	X: 305 – 2500	<input checked="" type="checkbox"/>
6			Y-Achse verläuft von 0 bis 65000	Y: 0 – 1	

Testprotokoll			Spektrometer App		
Test ID	0026		Durchgeführt von	Andreas	
Test Titel	Darstellung der „Reflectance“ Werte		Datum	08.03.2017	
Anforderung	6.0.003		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen					
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Erste Messung antippen	Erste Messung	Modus ist Raw	Modus ist auf Raw eingestellt	<input checked="" type="checkbox"/>
4	Modus Reflectance antippen	Erste Messung	Modus ist Reflectance	Modus ist auf Reflectance eingestellt	<input checked="" type="checkbox"/>
5			X Achse verläuft von 350 bis 2500	X: 305 – 2500	<input checked="" type="checkbox"/>
6			Y Achse verläuft von 0 bis 1.25	Y: 0 – 1.25	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0027		Durchgeführt von	Andreas	
Test Titel	Darstellung der „Transmittance“ Werte		Datum	08.03.2017	
Anforderung	6.0.004		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Prio 4: Nicht umgesetzt				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Erste Messung antippen	Erste Messung	Modus ist Raw	Modus ist auf Raw eingestellt	<input checked="" type="checkbox"/>
4	Modus Trans antippen	Erste Messung	Modus ist Transmittance	Modus ist auf Trans eingestellt	
5			X-Achse verläuft von 350 bis 2500	X: 305 – 2500	
6			Y-Achse verläuft von 0 bis 1.25	Y: 0 – 1.25	

Testprotokoll			Spektrometer App		
Test ID	0028		Durchgeführt von	Andreas	
Test Titel	Darstellung der „Absorbance“ Werte		Datum	08.03.2017	
Anforderung	6.0.005		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Prio 4: Nicht umgesetzt				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Erste Messung antippen	Erste Messung	Modus ist Raw	Modus ist auf Raw eingestellt	<input checked="" type="checkbox"/>
4	Modus Abs antippen	Erste Messung	Modus ist Absorbance	Modus ist auf Abs eingestellt	
5			X-Achse verläuft von 350 bis 2500	X: 305 – 2500	
6			Y-Achse verläuft von 0 bis 2	Y: 0 – 2	

Testprotokoll			Spektrometer App		
Test ID	0029		Durchgeführt von	Andreas	
Test Titel	Zoom der grafischen Darstellung		Datum	08.03.2017	
Anforderung	6.1.001		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Geste: Pinch				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>
3	Erste Messung antippen	Erste Messung	Modus ist Raw	Modus ist auf Raw eingestellt	<input checked="" type="checkbox"/>
4	Zoomen	Erste Messung	Das Diagramm wird vergrößert.	Das Diagramm wird vergrößert	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0030		Durchgeführt von	Andreas	
Test Titel	Anpassen der grafischen Darstellung		Datum	08.03.2017	
Anforderung	6.1.002		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Prio 4: Nicht umgesetzt				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>

Testprotokoll			Spektrometer App		
Test ID	0031		Durchgeführt von	Andreas	
Test Titel	Konfiguration der X- und Y-Achsen		Datum	08.03.2017	
Anforderung	6.1.003		Testgerät	Fieldspec 3 / Simulator	
Beschreibung					
Bedingungen					
Abhängigkeiten	Test 16 durchführen				
Bemerkungen	Prio 4: Nicht umgesetzt				
Nr.	Schritte	Testdaten	Erwartetes Resultat	Resultat	Erfüllt
1	Messungs-Tab öffnen		Messungen werden aufgelistet	Messungen werden aufgelistet	<input checked="" type="checkbox"/>
2	Ordner „Messungen“ aufrufen		Der Ordner Messungen wird angezeigt	Der Ordner Messungen wird angezeigt	<input checked="" type="checkbox"/>



ASD Inc.
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: support@asdi.com

Indico Version 8 File Format



Table of Contents

Table of Contents	i
Introduction	1
Spectrum File Header.....	3
Spectrum Data.....	5
Reference File Header	5
Reference Data.....	5
Classifier Data	5
Dependent Variables	6
Calibration Header.....	7
Base Calibration Data.....	7
Lamp Calibration Data.....	7
Fiber Optic Data.....	8
Audit Log.....	8
Signature	8

Introduction

Overview

The Indico file format is the format for storing both raw data as well as reference data. This format is created and used by the Indico, RS3 and 21CFR software. The following specification gives a detailed description of the structure for version 8 of this format.

Data Format

The ASD Indico file format is native to Windows and there for Intel processors, all data values are stored in Little-Endian (least significant byte first) order.

Variable length strings are stored in BSTR format. A BSTR (Basic string or binary string) is a string data type that is used by COM, Automation, and Interop functions. A BSTR is a composite data type that consists of a length prefix and the data string. The length prefix is a 4 byte integer that defines the length of the string. The data string is followed immediately after the length prefix.

Variable length arrays are stored in SAFEARRAY format.

File Structure

The Indico file layout consists of 12 sections: The following figure displays how these sections are laid out.

Basic Indico File Layout

Spectrum File Header

Spectrum Data

Reference File Header

Reference Data

Classifier Data

Dependent Variable Data

Calibration Header

Base Data

Lamp Data

Fiber Optic Data

Audit Log

Signature

Spectrum File Header

The spectrum file header section is the first section and consists of 484 bytes of data. The following table details the offset and data type format.

Offset	Size	Type	Description	Comment
	3	char	co[3];	// File Version - as6
3	157	char	comments[157];	// comment field
160	18	struct tm	when;	// time when spectrum was saved
178	1	byte	program_version;	// ver. of the program created this file.
				// major ver in upper nibble, min in lower
179	1	byte	file_version;	// spectrum file format version
180	1	byte	itime;	// Not used after v2.00
181	1	byte	dc_corr;	// 1 if DC subtracted, 0 if not
182	4	time_t (==long)	dc_time;	// Time of last dc, seconds since 1/1/1970
186	1	byte	data_type;	// see * _TYPE below
187	4	time_t (==long)	ref_time;	// Time of last wr, seconds since 1/1/1970
191	4	float	ch1_wavel;	// calibrated starting wavelength in nm
195	4	float	wavel_step;	// calibrated wavelength step in nm
199	1	byte	data_format;	// format of spectrum.
200	1	byte	old_dc_count;	// Num of DC measurements in the avg
201	1	byte	old_ref_count;	// Num of WR in the average
202	1	byte	old_sample_count;	// Num of spec samples in the avg
203	1	byte	application;	// Which application created APP_DATA
204	2	ushort	channels;	// Num of channels in the detector
206	128	APP_DATA	app_data;	// Application-specific data
334	56	GPS_DATA	gps_data;	// GPS position, course, etc.
390	4	ulong	it;	// The actual integration time in ms
394	2	int	fo;	// The fo attachment's view in degrees
396	2	int	dcc;	// The dark current correction value
398	2	uint	calibration;	// calibration series
400	2	uint	instrument_num;	// instrument number
402	4	float	ymin;	// setting of the y axis' min value
406	4	float	ymax;	// setting of the y axis' max value
410	4	float	xmin;	// setting of the x axis' min value
414	4	float	xmax;	// setting of the x axis' max value
418	2	uint	ip_numbits;	// instrument's dynamic range
420	1	byte	xmode;	// x axis mode. See * _XMODE
421	4	byte	flags[4];	// flags(0) = AVGFIX'ed
				// flags(1) see below
425	2	unsigned	dc_count;	// Num of DC measurements in the avg
427	2	unsigned	ref_count;	// Num of WR in the average
429	2	unsigned	sample_count;	// Num of spec samples in the avg
431	1	byte	instrument;	// Instrument type. See defs below
432	4	ulong	bulb;	// The id number of the cal bulb
436	2	uint	swir1_gain;	// gain setting for swir 1
438	2	uint	swir2_gain;	// gain setting for swir 2
440	2	uint	swir1_offset;	// offset setting for swir 1
442	2	uint	swir2_offset;	// offset setting for swir 2
444	4	float	splice1_wavelength;	// wavelength of VNIR and SWIR1 splice
448	4	float	splice2_wavelength;	// wavelength of SWIR1 and SWIR2 splice
452	27	float	SmartDetectorType	// Data from OL731 device
479	5	byte	spare[5];	// fill to 484 bytes

Definitions:

Spectrum data type (variable data type at byte offset 186):

```
#define RAW_TYPE          (byte) 0
#define REF_TYPE          (byte) 1
#define RAD_TYPE          (byte) 2
#define NOUNITS_TYPE      (byte) 3
#define IRRAD_TYPE        (byte) 4
#define QI_TYPE           (byte) 5
```



```
#define TRANS_TYPE      (byte) 6
#define UNKNOWN_TYPE    (byte) 7
#define ABS_TYPE        (byte) 8
```

Spectrum data format (variable data_format at byte offset 199):

```
#define FLOAT_FORMAT    (byte) 0
#define INTEGER_FORMAT  (byte) 1
#define DOUBLE_FORMAT   (byte) 2
#define UNKNOWN_FORMAT  (byte) 3
```

Instrument type that created spectrum (variable instrument at byte offset 431):

```
#define UNKNOWN_INSTRUMENT      (byte) 0
#define PSII_INSTRUMENT        (byte) 1
#define LSVNIR_INSTRUMENT      (byte) 2
#define FSVNIR_INSTRUMENT      (byte) 3
#define FSFR_INSTRUMENT        (byte) 4
#define FSNIR_INSTRUMENT       (byte) 5
#define CHEM_INSTRUMENT        (byte) 6
#define FSFR_UNATTENDED_INSTRUMENT (byte) 7
```

```
struct tm
{
    int    tm_sec;           // seconds [0,61]
    int    tm_min;          // minutes [0,59]
    int    tm_hour;         // hour [0,23]
    int    tm_mday;         // day of month [1,31]
    int    tm_mon;          // month of year [0,11]
    int    tm_year;         // years since 1900
    int    tm_wday;         // day of week [0,6] (Sunday = 0)
    int    tm_yday;         // day of year [0,365]
    int    tm_isdst;        // daylight savings flag
};
```

```
typedef long time_t;
```

APP_DATA - This is a 128 byte field that is used for storing results produced by various real-time processing routines.

```
struct GPS_DATA
{
    double    true_heading;
    double    speed;
    double    latitude, longitude;
    double    altitude;
    struct
    {
        unsigned havecomm : 1;
        unsigned terrain : 2;
        unsigned datum : 6;
        unsigned dist_sp_units : 2;
        unsigned alt_units : 2;
        unsigned mag_var : 2;
        unsigned nav : 1;
    } flags; // these are bit fields totaling to 2 bytes
    char    hardware_mode;
    time_t    timestamp;
    struct
    {
        unsigned corrected : 1;
        unsigned filler : 15;
    } flags2; // these are bit fields totaling to 2 bytes
    unsigned char    satellites[5];
    char    filler[2];
}

flags
flags(0)
flags(1)    vnir saturation = 1
            swirl1 saturation = 2
            swirl2 saturation = 3
            Tec1 alarm = 8
            Tec2 alarm = 16
```

```

struct SmartDetectorType
{
    int serial_number;           // Serial Number
    float Signal                 // Signal
    float dark                   // Dark Signal
    float ref                    // Ref Signal
    short Status                 // Smart Detector Status
    byte avg                     // Averaging
    float humid                  // Humidity
    float temp                   // Temperature
}

```

Spectrum Data

The spectrum data section consists of byte 485 to channels as defined in byte 204 in the spectrum file header. The following table details the offset and data type format.

Offset	Size	Type	Description	Comment
485	channels	double	Spectrum	// Spectrum data to size of channels

Reference File Header

The reference file header section consists of Spectrum Data Size + 1 to the size of Reference File Header. The following table details the offset and data type format.

Offset	Size	Type	Description	Comment
Spectrum Data size + 1	2	bool	ReferenceFlag	// Reference been taken
3	8	date	ReferenceTime	// Time Reference was taken
11	8	date	SpectrumTime	// Time Spectrum was taken
19	n	string	SpectrumDescription	// Description of Spectrum

Reference Data

The reference data section consists of Reference File Header size + 1 to channels as defined in byte 204 in the spectrum file header. The following table details the offset and data type format.

Offset	Size	Type	Description	Comment
Reference File Header size + 1	channels	double	Reference	// Reference data to size of channels

Classifier Data

The classifier data section consists of Reference Data size + 1 to size of Classifier Data. The following table details the offset and data type format of Classifier Data.

Offset	Size	Type	Description	Comment
Reference Data size + 1	1	byte	yCode	// Type of Classifier Data - 0=SAM, 1=GALACTIC, 2=CAMOPREDICT, 3=CAMOCCLASSIFY, 4=PCAZ, 5=INFOMETRIX
1	1	byte	yModelType	// Type of Model Quantify/Classify or both
2	n	String	sTitle	// Title of Classifier
n+1	n	String	sSubTitle	// SubTitle of Classifier
n+1	n	String	sProductName	// Product Name
n+1	n	String	sVendor	// Vender Name
n+1	n	String	sLotNumber	// LotNumber of Sample
n+1	n	String	sSample	// Sample Description
n+1	n	String	sModelName	// Model Description
n+1	n	String	sOperator	// Operator Name
n+1	n	String	sDateTime	// Date/time sample taken
n+1	n	String	sInstrument	// Instrument Name
n+1	n	String	sSerialNumber	// Serial Number of Instrument
n+1	n	String	sDisplayMode	// Display Mode
n+1	n	String	sComments	// Comments for sample
n+1	n	String	sUnits	// Units of Concentration
n+1	n	String	sFilename	// File Name for sample
n+1	n	String	sUserName	// User Name
n+1	n	String	sReserved1	// Reserved
n+1	n	String	sReserved2	// Reserved
n+1	n	String	sReserved3	// Reserved
n+1	n	String	sReserved4	// Reserved
n+1	2	integer	iConstituentCount	// Number of Constituents
n+3		ConstituentType	actConstituent()	// See definition below.

Definitions:

```

ConstituentType
{
    'Items in the Material Report
    ctConstituentName As String
    ctPassFail As String
    ctMDistance As Double
    ctMDistanceLimit As Double
    ctConcentration As Double
    ctConcentrationLimit As Double
    ctFRatio As Double
    ctResidual As Double
    ctResidualLimit As Double
    ctScores As Double
    ctScoresLimit As Double
    ctModelType As Long
    ctReserved1 As Double
    ctReserved2 As Double
}

```

Dependent Variables

The dependent variables section consists of Classifier Data size + 1 to size of Dependent Variables size. The following table details the offset and data type format of Dependent Variables.

Offset	Size	Type	Description	Comment
Classifier Data size + 1	1	bool	SaveDependentVariables	// Has reference been taken
1	2	integer	DependentVariableCount	// Number of dependent variables
4	n	String	DependentVariableLabels()	// Names of dependents variables
n+1	4	float	DependentVariables()	// Values of dependent variables

Calibration Header

The calibration header defines the calibration data to follow. The count field defines the number of calibration buffers contained in the file. The CalBuffer holds data about each calibration buffer. The following table details the offset and data type format of Calibration Header.

Offset	Size	Type	Description	Comment
Dependent Variables size + 1	1 byte		Count	// Number of calibration buffers in the file.
1	29	CalBuffer	Structure for each calibration buffer	// Defines the Type, Name, Integration Time and Gains of buffer

Definitions:

```
typedef enum _CAL_TYPE
{
    ABS,                // Absolute Reflectance File
    BSE,                // Base File
    LMP,                // Lamp File
    FO,                 // Fiber Optic File
} CALIBRATION_TYPE;

struct
{
    byte    cbType           // ABS, BSE, LMP or FO
    char    cbName[20]      // Name of file
    long    cbIT            // Integration Time in ms of buffer
    int     cbSwirlGain      // Swirl Gain of buffer
    int     cbSwirl2Gain     // Swirl2 Gain of buffer
} CAL_BUFFER;
```

Base Calibration Data

This section consists of either absolute reflectance or base calibration data. The cbType field of the calibration header for the first element will define the type. The following table details the offset and data type format.

Offset	Size	Type	Description	Comment
Calibration Header size + 1	channels	double	Absolute Reflectance or Base file	// data to size of channels defined in the Spectrum Header.

Lamp Calibration Data

This section consists the lamp data. The cbType field of the calibration header for the second element will define the type. The following table details the offset and data type format.

Offset	Size	Type	Description	Comment
Base calibration data size + 1	channels	double	Lamp file	// data to size of channels defined in the Spectrum Header.

Fiber Optic Data

This section consists the fiber optic data. The cbType field of the Calibration header for the second element will define the type. The following table details the offset and data type format.

Offset	Size	Type	Description	Comment
Lamp calibration data size + 1	channels	double	Fiber optic file	// data to size of channels defined in the Spectrum Header. The type of fiber optic is defined in the fo field in the Spectrum Header.

Audit Log

This section defines the Audit Log for each signature event. The log event is defined by the <Audit_Event> and </Audit_Event> tags. Within the audit event are tags that define the log event. Below is a sample of an audit log.

Offset	Size	Type	Description	Comment
0	4	long	Count	// Number of log events in the string array .
4	n	String Array	AuditEvents	// String array for each audit event

Definitions:

```

<Audit_Event>
<Audit_Application>Indico Pro</Audit_Application>
<Audit_AppVersion> 6.0</Audit_AppVersion>
<Audit_Name>Bryon Bending</Audit_Name>
<Audit_Login>\\ASDI\bryon.bending</Audit_Login>
<Audit_Time>2009/12/12 14:11:22 GMT</Audit_Time>
<Audit_Source>c:\ASD\Data\Sample.asd</Audit_Source>
<Audit_Function>Initial Collection</Audit_Function>
<Audit_Notes>Sample prepared and collected following SOP # TG101</Audit_Notes>
</Audit_Event>

<Audit_Event>
<Audit_Application>Spectral Viewer</Audit_Application>
<Audit_AppVersion> 1.0</Audit_AppVersion>
<Audit_Name>Don Campbell</Audit_Name>
<Audit_Login>\\ASDI\don.campbell</Audit_Login>
<Audit_Time>2009/12/12 15:11:22 UTC</Audit_Time>
<Audit_Source>c:\ASD\Data\Sample.asd</Audit_Source>
<Audit_Function>Approval</Audit_Function>
<Audit_Notes>Sample approved</Audit_Notes>
</Audit_Event>

```

Signature

The section defines the electronic signature of the file. The signature details the user who signed the file, when the file was signed, source file and a reason for signing. The electronic signature uses asymmetric cryptography. Asymmetric Cryptography uses both a private and public key. The private key is used to encrypt the record, while the public key is used to decrypt the record. The signature in the record will consist of the private key and a hash of the record. To detect an altered record a user must compute the hash of the record and compare it to the signature with the signer's public key.

Offset	Size	Type	Description	Comment
0	1	byte	Signed	// 0 – Unsigned 1 - Signed
2	8	date	SignatureTime	// Date and Time File was signed. Value is stored in UTC time.
11	n	string	UserDomain	// Users Login domain
n+1	n	string	UserLogin	// Users Login
n+1	n	string	UserName	// Users Name
n+1	n	String	Source	// Source file at time of signature
n+1	n	string	Reason	// Reason for signature
n+1	n	string	Notes	// Additional notes for the signature
n+1	n	string	PublicKey	// User Public Key
n+1	128	String(128)	Signature	// User Signature – Hash of Record + Private Key



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

TCPServer Developers Guide

Revision E

Table of Contents

Getting Started	1
Network Configuration	1
What's New	1
TCPServer API Documentation.....	3
A – Acquire data	8
A,1,x,x – Set sample count and Acquire data	10
A,2,x – Set Integration time and Acquires data	12
A,3,x,x – Set Swir1 Gain and Offset and Acquires data	14
A,4,x,x – Set Swir2 Gain and Offset and Acquires data	16
A,5,x – Toggle the shutter and Acquires data	18
ABORT – Abort command	20
ERASE – Clears the flash	21
IC,0,1,x – Instrument Gain Control for SWIR1	22
IC,0,2,x – Instrument Offset Control for SWIR1	24
IC,1,1,x – Instrument Gain Control for SWIR2	26
IC,1,2,x – Instrument Offset Control for SWIR2	28
IC,2,0,x – Instrument Integration Time Control for VNIR	30
IC,2,3,x – Instrument Shutter Control for VNIR	32
IC,2,4,0 – Instrument Trigger Reset	34
INIT,0,x – Gets parameter from flash	35
INIT,1,x,x – Adds a parameter to flash	36
INIT,2,x,x – Changes a parameter stored in flash	37
OPT,1 – Optimize VNIR detector	38
OPT,2 – Optimize SWIR1 detector	40
OPT,3 – Optimize VNIR and SWIR1 detectors	42
OPT,4 – Optimize SWIR2 detector	44
OPT,5 – Optimize VNIR and SWIR2 detectors	46
OPT,6 – Optimize SWIR1 and SWIR2 detectors	48
OPT,7 – Optimize VNIR, SWIR1 and SWIR2 detectors	50
RESTORE,x – Loads the flash into RAM	52
SAVE – Saves the values in RAM to flash	55
V – Version	56
Dark Current Collection	57
Writing a TCP Client	60
Making and closing a connection	60
Reading the starting and ending wavelength	61
Optimize	61
Acquiring data	61
Displaying a Dark Corrected Spectrum	63
Displaying a Reflectance Spectrum	64
Normalizing a Spectrum	65
Support	67

Getting Started

This guide will provide an overview on how to install, configure and write a sample application to communicate with your ASD Ethernet instrument.

Network Configuration

To communicate through the Ethernet or Wireless interface, configure the host computer network adapter's Internet Protocol Version 4 (TCP/IPv4) to "Obtain an IP address automatically". The IP address for the ASD Instrument is set to 169.254.1.11.

What's New

Version 3.0

- Integrate 802.11 n wireless interface.

Version 2.2

- Integrate 802.11 g wireless interface.

Version 1.6

- Add dark current floor check and update vnir drift values.

Version 1.5

- Added AB Equal interface to A command.
- New Interpolation routines.

Version 1.4

- Added support for Trigger feedback.

Version 1.3

- Added header structure to Acquire command
- Added wireless capability

Version 1.2

- Added ABORT command
- Added IC command
- Added V command
- Added OPT command
- Added support Vnir only instrument type.
- Added support for Vnir/Swir1 instrument type.
- Added support for Vnir/Swir2 instrument type.
- Added support for Swir1/Swir2 instrument type.
- Added support for Swir1 only instrument type.
- Added support for Swir2 only instrument type.



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

Version 1.1

Released for Full Range instruments only.

Version 1.0

Initial Release

TCPServer API Documentation

The command interface is a comma delimited character string. The total number of parameters in the command structure is 4. An example command may look like the following: "A,1,10". The first parameter is the command. Valid entries are defined in Table 1. The second parameter is the command type for the specified command. The third and fourth parameters in the command string are parameters for the command type. Valid entries are defined in Table 2. Table 3 defines the return structures of the requested command.

Table 1 Commands

Command	Description
A	Collect interpolated data.
ABORT	Aborts "A" and "OPT" commands
ERASE	Clears the contents of the flash.
IC	Instrument control command
INIT	Get, add or change ini file settings in the flash.
OPT	Optimize the instrument
RESTORE	Get and return the contents of the flash.
SAVE	Save ini file settings to the flash.
V	Version of firmware

Table 2 Command Type and Parameters

Param1	Param2	Param3	Param4	Description
A	<None>	<None>	<None>	Reset, then Acquire.
	1	1-32767	0-3	Set Sample Count. Example: "A,1,10,0" Sets the sample count to 10 with equal A and B scans.
	2	-1 - 15	<None>	Set Integration Time. Requires a third parameter: -1 - 15. This third parameter is the index value of the integration time. Example: "A,2,0" Sets the Vnir integration time to 17 ms.
	3	0-4096	0-4096	Set Gain and Offset of Swir1. Requires a third and fourth parameter. The third parameter is the Gain value to set. The fourth parameter is the Offset value to set. Example: "A,3,500,2048" Sets Swir1 Gain to 500 and Offset to 2048
	4	0-4096	0-4096	Set Gain and Offset of Swir2. Requires a third and fourth parameter. The third parameter is the Gain value to set. The fourth parameter is the Offset value to set. Example: "A,4,500,2048" Sets Swir2 Gain to 500 and Offset to 2048
	5	0-1	<None>	Toggle the shutter. Requires a third parameter. 0 to open the shutter. 1 to close the shutter. Example: "A,5,0" Open shutter. "A,5,1" Close shutter.
ABORT	<None>	<None>	<None>	Aborts current "A" and "OPT" command
ERASE	<None>	<None>	<None>	Clears the contents of the flash Example: "ERASE"
IC	0 - 2	0 - 4	-1 - 4096	Param2 values 0 – Swir1 1 – Swir2 2 – Vnir Param3 values 0 – Integration Time. Valid param4 values -1 - 15 1 – Gain Valid param4 values 0-4096 2 – Offset Valid param4 values 0-4096

				3 – Shutter Valid param4 values 0-1 4 – Trigger Valid param4 values 0 Param4 values – 0 - 4096 Example: “IC,2,0” Sets Vnir Integration Time to 17 ms “IC,0,1,500” Sets Swir1 Gain to 500 “IC,1,2,2048” Sets Swir2 Offset to 2048 “IC,2,3,1” Closes the Vnir shutter. “IC,2,3,0” Open the Vnir shutter.
INIT	0	30 char	<None>	Get value from flash. Requires a third parameter. The third parameter is the character string of a name of the value to get. ie. “SerialNumber” Example: “INIT,0,SerialNumber” gets the Serial Number from flash.
	1	30 char	double	Add a new to flash. Requires a third and fourth parameter. The third parameter is a character string of the name of the value ie. “SerialNumber. The fourth parameter is the value to set ie. “4012” Example: “INIT,1,SerialNumber,4012” Adds a Serial Number with a value of 4012 to the flash.
	2	30 char	double	Change a flash value. Requires a third and fourth parameter. The third parameter is a character string of the name of the value ie. “SerialNumber. The fourth parameter is the value to set ie. “4012” Example: “INIT,2,SerialNumber,4028” Changes the SerialNumber key to 4028.
OPT	1	<None>	<None>	Optimize VNIR device (BITMASK = 0x01). Upon successful completion of command, instrument values are set to optimized value(s).
	2	<None>	<None>	Optimize SWIR1 device (BITMASK = 0x02). Upon successful completion of command, instrument values are set to optimized value(s).
	3	<None>	<None>	Optimize VNIR and SWIR1 devices. Upon successful completion of command, instrument values are set to optimized value(s).
	4	<None>	<None>	Optimize SWIR2 device (BITMASK = 0x04). Upon successful completion of command, instrument values are set to optimized value(s).
	5	<None>	<None>	Optimize VNIR and SWIR2 device. Upon successful completion of command, instrument values are set to optimized value(s).
	6	<None>	<None>	Optimize SWIR1 and SWIR2 devices. Upon successful completion of command, instrument values are set to optimized value(s).
	7	<None>	<None>	Optimize VNIR, SWIR1 and SWIR2 devices. Upon successful completion of command, instrument values are set to optimized value(s).
RESTORE	0 - 1	<None>	<None>	Get and return the values from flash. Param2 0 - Loads the INI only 1 - Loads the INI and builds the calibration arrays. Example: “RESTORE,1”
SAVE	<None>	<None>	<None>	Save the current ini settings to flash. Example: “SAVE”
V	<None>	<None>	<None>	Returns the version of the TCP Server

Table 3 Return Packet structure.

Command	Return packet
A	<pre> // FRSpectrumHeader struct Vnir_Header { int IT; // Integration Time of vnir. int scans; // Number of scans in vnir region int max_channel; // Maximum DN value of vnir region int min_channel; // Minimum DN value of vnir region. int saturation; // Saturation Alarm 0 – no saturation 1 - saturation int shutter; // Shutter status 0 – Open 1 - Closed int drift; // Drift average value for defined drift channels int dark_subtracted; // Dark subtracted 0 – No 1 - Yes int reserved[8]; }; struct Swir_Header { int tec_status; // Tec Alarm 0 – No Alarm 1 or 2 Alarm int tec_current; // DN value of TEC controller int max_channel; // Maximum DN value of swir region int min_channel; // Minimum DN value of swir region int saturation; // Saturation Alarm 0 – no saturation 1 - saturation int A_Scans; // Number of A Scans in swir region int B_Scans; // Number of B Scans in swir region int dark_current; // Averaged Dark Current value int gain; // gain value of swir region int offset; // offset value of swir region int scansize1; // A Scan - Number of channels before encoder index // B Scan – Number of channels after encoder index int scansize2; // A Scan - Number of channels after encoder index // B Scan – Number of channels before encoder index int dark_subtracted; // Dark subtracted 0 – No 1 - Yes int reserved[3]; }; struct SpectrumHeader { int header; // Header code for Acquire int errbyte; // Error code for Acquire int sample_count; // Sample count of spectrum int trigger; // Trigger 0 – off 1 - on int voltage; // DN value of voltage. int current; // DN value of current. int temperature; // DN value of inside temperature. int motor_current; // DN value of motor current. int instrument_hours; // Number of runtime hours since last calibration. int instrument_minutes; // Number of runtime minutes since last calibration. int instrument_type; // 1 – 13 see version command for values int AB; // 0 – 3 see A command for value int reserved[4]; Vnir_Header v_header; // Vnir structure Swir_Header s1_header; // Swir1 structure Swir_Header s2_header; // Swir2 structure }; // Interpolated structure to return for Full Range Instrument // Applies to the FR_TCPServer firmware // struct FRInterpSpecStruct { SpectrumHeader FRSpectrumHeader; //256 bytes (64 words) float SpecBuffer [2151]; }; // // Interpolated structure to return for Vnir Spectrometers // Applies to the V_TCPServer firmware </pre>

Command	Return packet
	<pre>// struct VInterpSpecStruct { SpectrumHeader VSpectrumHeader; float SpecBuffer [701]; };// // Interpolated structure to return for Swir1 Swir2 Spectrometers // Applies to the S1S2_TCPServer firmware // struct S1S2InterpSpecStruct { SpectrumHeader S1S2SpectrumHeader; float SpecBuffer [1502]; };// // Interpolated structure to return for Swir1 Spectrometers // Applies to the S1_TCPServer firmware // struct S1InterpSpecStruct { SpectrumHeader S1SpectrumHeader; float SpecBuffer [801]; }; // // Interpolated structure to return for Swir2 Spectrometers // Applies to the S2_TCPServer firmware // struct S2InterpSpecStruct { SpectrumHeader S2SpectrumHeader; float SpecBuffer [701]; }; // // Interpolated structure to return for Vnir/Swir1 Spectrometers // Applies to the VS1_TCPServer firmware // struct VS1InterpSpecStruct { SpectrumHeader VS1SpectrumHeader; float SpecBuffer [1502]; }; // // Interpolated structure to return for Vnir/Swir2 Spectrometers // Applies to the VS2_TCPServer firmware // struct VS2InterpSpecStruct { SpectrumHeader VS2SpectrumHeader; float SpecBuffer [1402]; };</pre>
ABORT	<pre>Struct ParamStruct { int header; int errbyte; char name[30]; double value; int count; }</pre>
ERASE	<pre>struct InitStruct { int header; //header type used in TCP transfer. int errbyte; //error code char name [MAX_PARAMETERS][30]; //space for 200 entries with 30 character names double value [MAX_PARAMETERS]; //corresponding data values for the 200 entries int count; //The number of used entries</pre>

Command	Return packet
	<pre> int verify; //the checksum }; </pre>
IC	<pre> struct InstrumentControlStruct { int header; // header type used in TCP transfer int errbyte; // error code int detector; // Detector number – 0 swir1, 1 swir2, 2 vnir int cmdType; // Command Type 0 IT, 1 Gain, 2 Offset, 3 Shutter, 4 Trigger int value; // Value issues 0 - 4096 }; </pre>
INIT	<pre> struct ParamStruct { int header; //header type used in TCP transfer. int errbyte; //error code char name[30]; //space for 200 entries with 30 character names double value; //corresponding data values for the 200 entries int count; //number of entries used }; </pre>
OPT	<pre> struct OptimizeStruct { int header; //header type used in TCP transfer. int errbyte; //error code int itime; //optimized integration time int gain[2]; //optimized gain for 2 SWIRs int offset[2]; //optimized offset for 2 SWIRs }; </pre>
RESTORE	<pre> struct InitStruct { int header; //header type used in TCP transfer. int errbyte; //error code char name[MAX_PARAMETERS][30]; //space for 200 entries with 30 character names double value[MAX_PARAMETERS]; //corresponding data values for the 200 entries int count; //The number of used entries int verify; //the checksum }; </pre>
SAVE	<pre> struct InitStruct { int header; //header type used in TCP transfer. int errbyte; //error code char name[MAX_PARAMETERS][30]; //space for 200 entries with 30 character names double value[MAX_PARAMETERS]; //corresponding data values for the 200 entries int count; //The number of used entries int verify; //the checksum }; </pre>
V	<pre> struct VersionStruct { int header; // header type used in TCP transfer. int errbyte; // error code char version[30]; // 30 character Version and build double value; // Version number int type; // Type of instrument 1-Vnir, 4-Swir1, 5-Vnir/Swir1 }; // 8-Siwr2, 9-Vnir/Swir2 // 12-Swir1/Swir2, 13-Vnir/Swir1/Swir2 </pre>

A – Acquire data

Description:

This command resets the detectors then collects and interpolates data at the current instrument settings.

Note: This command requires the instrument ini and calibration arrays to be loaded into the flash. See RESTORE for Details.

Parameters

Param1
“A” Identifies Acquire command.

Param2
Not Used

Param3
Not Used

Param4
Not Used

Returns

```
Struct FRInterpSpecStruct
{
    SpectrumHeader FRSpectrumHeader;
    float SpecBuffer[2151];
}
```

header

H_NO_ERROR	100
H_COLLECT_ERROR	200
H_COLLECT_NOT_LOADED	300
H_RESET_ERROR	600
H_INTERPOLATE_ERROR	700

errbyte

NO_ERROR	0
NOT_READY	-1
NO_INDEX_MARKS	-2
TOO_MANY_ZEROS	-3
SCANSIZE_ERROR	-4
VNIR_TIMEOUT	-10
SWIR_TIMEOUT	-11
VNIR_NOT_READY	-12
SWIR1_NOT_READY	-13
SWIR2_NOT_READY	-14
ABORT_ERROR	-18
VNIR_INTERP_ERROR	-20
SWIR1_INTERP_ERROR	-21
SWIR2_INTERP_ERROR	-22



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

SpecBuffer

Interpolated spectrum buffer.

See Table 3 for additional information on the return structures and header definition.

Example

“A”

Collects and interpolates data at the currently set sample count, integration time, gain and offsets.

A,1,x,x – Set sample count and Acquire data

Description:

This command sets the sample count, resets the detectors, collects and interpolates spectrum data.

Note: This command requires the instrument ini and calibration arrays to be loaded into the flash. See RESTORE for Details.

Parameters

<i>Param1</i>	"A"	Identifies the Acquire command.
<i>Param2</i>	1	Set Sample Count command type.
<i>Param3</i>	1-32767	Sample count
<i>Param4</i>	0 – 3	Scan Type 0 – (Default) A and B Even spectrum averaging 1 – A only 2 – B only 3 – A and B.

Returns

```
Struct FRInterpSpecStruct
{
    SpectrumHeader FRSpectrumHeader;
    float SpecBuffer[2151];
}
```

header

H_NO_ERROR	100
H_COLLECT_ERROR	200
H_COLLECT_NOT_LOADED	300
H_RESET_ERROR	600
H_INTERPOLATE_ERROR	700

errbyte

NO_ERROR	0
NOT_READY	-1
NO_INDEX_MARKS	-2
TOO_MANY_ZEROS	-3
SCANSIZE_ERROR	-4
VNIR_TIMEOUT	-10
SWIR_TIMEOUT	-11
VNIR_NOT_READY	-12
SWIR1_NOT_READY	-13
SWIR2_NOT_READY	-14
ABORT_ERROR	-18
VNIR_INTERP_ERROR	-20



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

SWIR1_INTERP_ERROR	-21
SWIR2_INTERP_ERROR	-22

SpecBuffer

Interpolated spectrum buffer.

See Table 3 for additional information on the return structures and header definition.

Example

“A,1,10” Sets the sample count to 10 and returns interpolated data.

A,2,x – Set Integration time and Acquires data

Description:

This command sets the integration time, resets the detectors, collects and interpolates spectrum data.

Note: This command requires the instrument ini and calibration arrays to be loaded into the flash. See RESTORE for Details.

Parameters

<i>Param1</i>	"A"	Identifies the Acquire command.
<i>Param2</i>	2	Set Integration Time command type.
<i>Param3</i>	Index	Integration Time
	-1	8.5ms
	0	17ms
	1	34ms
	2	68ms
	3	136ms
	4	272ms
	5	544ms
	6	1.09sec
	7	2.18sec
	8	4.35sec
	9	8.70sec
	10	17.41sec
	11	34.82sec
	12	1.16min
	13	2.32min
	14	4.64min
	15	9.28min
<i>Param4</i>	Not Used	

Returns

```
Struct FRInterpSpecStruct
{
    SpectrumHeader FRSpectrumHeader;
    float SpecBuffer[2151];
}
```

<i>header</i>	H_NO_ERROR	100
	H_COLLECT_ERROR	200
	H_COLLECT_NOT_LOADED	300
	H_RESET_ERROR	600
	H_INTERPOLATE_ERROR	700

errbyte

NO_ERROR	0
NOT_READY	-1
NO_INDEX_MARKS	-2
TOO_MANY_ZEROS	-3
SCANSIZE_ERROR	-4
VNIR_TIMEOUT	-10
SWIR_TIMEOUT	-11
VNIR_NOT_READY	-12
SWIR1_NOT_READY	-13
SWIR2_NOT_READY	-14
ABORT_ERROR	-18
VNIR_INTERP_ERROR	-20
SWIR1_INTERP_ERROR	-21
SWIR2_INTERP_ERROR	-22

SpecBuffer

Interpolated spectrum buffer.

See Table 3 for additional information on the return structures and header definition.

Example

“A,2,0” Sets the integration time to 17ms.

A,3,x,x – Set Swir1 Gain and Offset and Acquires data

Description:

This command sets the gain and offset for swir1, resets the detectors, collects and interpolates spectrum data.

Note: This command requires the instrument ini and calibration arrays to be loaded into the flash. See RESTORE for Details.

Parameters

<i>Param1</i>	"A"	Identifies the Acquires command.
<i>Param2</i>	3	Set Gain and Offset for swir1 command type.
<i>Param3</i>	0- 4096	Gain value
<i>Param4</i>	0-4096	Offset value

Returns

Struct FRInterpSpecStruct

```
{
    SpectrumHeader FRSpectrumHeader;
    float SpecBuffer[2151];
}
```

<i>header</i>	H_NO_ERROR	100
	H_COLLECT_ERROR	200
	H_COLLECT_NOT_LOADED	300
	H_RESET_ERROR	600
	H_INTERPOLATE_ERROR	700

<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	NO_INDEX_MARKS	-2
	TOO_MANY_ZEROS	-3
	SCANSIZE_ERROR	-4
	VNIR_TIMEOUT	-10
	SWIR_TIMEOUT	-11
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	ABORT_ERROR	-18
	VNIR_INTERP_ERROR	-20
	SWIR1_INTERP_ERROR	-21
	SWIR2_INTERP_ERROR	-22



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

SpecBuffer

Interpolated spectrum buffer.

See Table 3 for additional information on the return structures and header definition.

Example

“A,3,500,2048” Sets the Gain of Swir1 to 500 and Offset to 2048.

A,4,x,x – Set Swir2 Gain and Offset and Acquires data

Description:

This command sets the gain and offset for swir2, resets the detectors, collects and interpolates spectrum data.

Note: This command requires the instrument ini and calibration arrays to be loaded into the flash. See RESTORE for Details.

Parameters

<i>Param1</i>		
	"A"	Identifies the Acquire command.
<i>Param2</i>		
	4	Set Gain and Offset for swir2 command type.
<i>Param3</i>		
	0- 4096	Gain value
<i>Param4</i>		
	0-4096	Offset value

Returns

Struct FRInterpSpecStruct

```
{
    SpectrumHeader FRSpectrumHeader;
    float SpecBuffer[2151];
}
```

header

H_NO_ERROR	100
H_COLLECT_ERROR	200
H_COLLECT_NOT_LOADED	300
H_RESET_ERROR	600
H_INTERPOLATE_ERROR	700

errbyte

NO_ERROR	0
NOT_READY	-1
NO_INDEX_MARKS	-2
TOO_MANY_ZEROS	-3
SCANSIZE_ERROR	-4
VNIR_TIMEOUT	-10
SWIR_TIMEOUT	-11
VNIR_NOT_READY	-12
SWIR1_NOT_READY	-13
SWIR2_NOT_READY	-14
ABORT_ERROR	-18
VNIR_INTERP_ERROR	-20
SWIR1_INTERP_ERROR	-21
SWIR2_INTERP_ERROR	-22



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

SpecBuffer

Interpolated spectrum buffer.

See Table 3 for additional information on the return structures and header definition.

Example

“A,4,500,2048” Sets the Gain of Swir2 to 500 and Offset to 2048.

A,5,x – Toggle the shutter and Acquires data

Description:

This command toggles the shutter for the vnir, resets the detectors, collects and interpolates spectrum data.

Note: This command requires the instrument ini and calibration arrays to be loaded into the flash. See RESTORE for Details.

Parameters

<i>Param1</i>	"A"	Identifies the Acquire command.
<i>Param2</i>	5	Toggle the shutter.
<i>Param3</i>	0	Open the shutter
	1	Close the shutter
<i>Param4</i>	Not Used	

Returns

```
Struct FRInterpSpecStruct
{
    SpectrumHeader FRSpectrumHeader;
    float SpecBuffer[2151];
}
```

<i>header</i>	H_NO_ERROR	100
	H_COLLECT_ERROR	200
	H_COLLECT_NOT_LOADED	300
	H_RESET_ERROR	600
	H_INTERPOLATE_ERROR	700

<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	NO_INDEX_MARKS	-2
	TOO_MANY_ZEROS	-3
	SCANSIZE_ERROR	-4
	VNIR_TIMEOUT	-10
	SWIR_TIMEOUT	-11
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	ABORT_ERROR	-18
	VNIR_INTERP_ERROR	-20
	SWIR1_INTERP_ERROR	-21
	SWIR2_INTERP_ERROR	-22



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

SpecBuffer

Interpolated spectrum buffer.

See Table 3 for additional information on the return structures and header definition.

Example

“A,5,0” Opens the Shutter

“A,5,1” Closes the Shutter

ABORT – Abort command

Description:

This command Aborts the current “A” and “OPT” commands in the command queue.

Parameters

Param1
“ABORT” Identifies the Abort command.

Param2
Not Used.

Param3
Not Used.

Param4
Not Used.

Returns

```
Struct ParamStruct
{
    int header;
    int errbyte;
    char name[30];
    double value;
    int count;
}
```

header
H_NO_ERROR 100

errbyte
NO_ERROR 0

name
“ABORT”

value
Not Used.

count
Not Used.

Example

“ABORT” Aborts the current “A” and “OPT” commands in the command queue.

ERASE – Clears the flash

Description:

This command clears the flash.

Parameters

<i>Param1</i>	“ERASE”	Identifies the ERASE command.
<i>Param2</i>	Not Used.	
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct InitStruct

```
{
    int header;
    int errbyte;
    char name[200][30];
    double value[200];
    int count;
    int verify;
}
```

<i>header</i>	H_NO_ERROR	100
	H_FLASH_ERROR	500
<i>errbyte</i>	NO_ERROR	0
<i>name</i>	Space for 200 entries with 30 character names.	
<i>value</i>	Corresponding data value for 200 entries.	
<i>count</i>	The number of used entries.	
<i>verify</i>	The checksum value.	

Example

“ERASE” Clears the flash.

IC,0,1,x – Instrument Gain Control for SWIR1

Description:

This command sets the gain value for SWIR1.

Parameters

<i>Param1</i>	"IC"	Identifies the Instrument Control command.
<i>Param2</i>	0	SWIR1 Detector
<i>Param3</i>	1	Gain control
<i>Param4</i>	0-4096	Gain value to set

Returns

Struct InstrumentControlStruct

{	int header;	
	int errbyte;	
	int detector;	
	int cmdType;	
	int value;	
}		
<i>header</i>	H_NO_ERROR	100
	H_INSTRUMENT_CONTROL_ERROR	900
<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	PARAM_ERROR	-19
<i>detector</i>	0	SWIR1
	1	SWIR2
	2	VNIR
<i>cmdType</i>	0	Integration Time
	1	Gain
	2	Offset
	3	Shutter
<i>values</i>	0 - 4096	



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

Example

“IC,0,1,500”

Sets the Gain to 500 for SWIR1.

IC,0,2,x – Instrument Offset Control for SWIR1

Description:

This command sets the offset value for SWIR1.

Parameters

<i>Param1</i>	"IC"	Identifies the Instrument Control command.
<i>Param2</i>	0	SWIR1 Detector
<i>Param3</i>	2	Offset control
<i>Param4</i>	0-4096	Offset value to set

Returns

Struct InstrumentControlStruct

{	int header;	
	int errbyte;	
	int detector;	
	int cmdType;	
	int value;	
}		
<i>header</i>	H_NO_ERROR	100
	H_INSTRUMENT_CONTROL_ERROR	900
<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	PARAM_ERROR	-19
<i>detector</i>	0	SWIR1
	1	SWIR2
	3	VNIR
<i>cmdType</i>	0	Integration Time
	1	Gain
	2	Offset
	3	Shutter
<i>values</i>	0 - 4096	



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

Example

“IC,0,2,2048”

Sets the Offset to 2048 for SWIR1.

IC,1,1,x – Instrument Gain Control for SWIR2

Description:

This command sets the gain value for SWIR2.

Parameters

<i>Param1</i>	"IC"	Identifies the Instrument Control command.
<i>Param2</i>	1	SWIR2 Detector
<i>Param3</i>	1	Gain control
<i>Param4</i>	0-4096	Gain value to set

Returns

Struct InstrumentControlStruct

{	int header;	
	int errbyte;	
	int detector;	
	int cmdType;	
	int value;	
}		
<i>header</i>	H_NO_ERROR	100
	H_INSTRUMENT_CONTROL_ERROR	900
<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	PARAM_ERROR	-19
<i>detector</i>	0	SWIR1
	1	SWIR2
	2	VNIR
<i>cmdType</i>	0	Integration Time
	1	Gain
	2	Offset
	3	Shutter
<i>values</i>	0 - 4096	



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

Example

“IC,1,1,500”

Sets the Gain to 500 for SWIR2.

IC,1,2,x – Instrument Offset Control for SWIR2

Description:

This command sets the offset value for SWIR2.

Parameters

<i>Param1</i>	"IC"	Identifies the Instrument Control command.
<i>Param2</i>	1	SWIR2 Detector
<i>Param3</i>	2	Offset control
<i>Param4</i>	0-4096	Offset value to set

Returns

Struct InstrumentControlStruct

```
{
    int header;
    int errbyte;
    int detector;
    int cmdType;
    int value;
}
```

<i>header</i>	H_NO_ERROR	100
	H_INSTRUMENT_CONTROL_ERROR	900
<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	PARAM_ERROR	-19
<i>detector</i>	0	SWIR1
	1	SWIR2
	2	VNIR
<i>cmdType</i>	0	Integration Time
	1	Gain
	2	Offset
	3	Shutter
<i>values</i>	0 - 4096	



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

Example

“IC,1,2,2048”

Sets the Offset to 2048 for SWIR2.

IC,2,0,x – Instrument Integration Time Control for VNIR

Description:

This command sets the integration time value index for VNIR.

Parameters

Param1
"IC" Identifies the Instrument Control command.

Param2
2 VNIR Detector

Param3
0 Integration Time control

Param4

Index	Integration Time
-1	8.5ms
0	17ms
1	34ms
2	68ms
3	136ms
4	272ms
5	544ms
6	1.09sec
7	2.18sec
8	4.35sec
9	8.70sec
10	17.41sec
11	34.82sec
12	1.16min
13	2.32min
14	4.64min
15	9.28min

Returns

```
Struct InstrumentControlStruct
{
    int header;
    int errbyte;
    int detector;
    int cmdType;
    int value;
}
```

header

H_NO_ERROR	100
H_INSTRUMENT_CONTROL_ERROR	900

errbyte

NO_ERROR	0
----------	---



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

NOT_READY	-1
VNIR_NOT_READY	-12
SWIR1_NOT_READY	-13
SWIR2_NOT_READY	-14
PARAM_ERROR	-19

detector

0	SWIR1
1	SWIR2
2	VNIR

cmdType

0	Integration Time
1	Gain
2	Offset
3	Shutter

values

-1 - 15

Example

“IC,2,0,0” Sets the integration time index to 17ms for the VNIR detector.

IC,2,3,x – Instrument Shutter Control for VNIR

Description:

This command toggles the shutter for VNIR.

Parameters

<i>Param1</i>		
	"IC"	Identifies the Instrument Control command.
<i>Param2</i>		
	2	VNIR Detector
<i>Param3</i>		
	3	Shutter control command
<i>Param4</i>		
	0	Open shutter
	1	Close shutter

Returns

Struct InstrumentControlStruct

{		
	int header;	
	int errbyte;	
	int detector;	
	int cmdType;	
	int value;	
}		
<i>header</i>		
	H_NO_ERROR	100
	H_INSTRUMENT_CONTROL_ERROR	900
<i>errbyte</i>		
	NO_ERROR	0
	NOT_READY	-1
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	PARAM_ERROR	-19
<i>detector</i>		
	0 SWIR1	
	1 SWIR2	
	2 VNIR	
<i>cmdType</i>		
	0 Integration Time	
	1 Gain	
	2 Offset	
	3 Shutter	
<i>values</i>		
	0 - 4096	



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

Example

“IC,2,3,0”

Opens the shutter for the VNIR detector.

“IC,2,3,1”

Closes the shutter for the VNIR detector.

IC,2,4,0 – Instrument Trigger Reset

Description:

This command resets the Trigger for activation. When the trigger is pressed, the LEDs turn on and the instrument sends a “Trigger” character string to the client. The trigger becomes inactive until it has been reset. Use this command to turn off the LEDs and reactivate the trigger.

Parameters

<i>Param1</i>	“IC”	Identifies the Instrument Control command.
<i>Param2</i>	2	VNIR Detector
<i>Param3</i>	4	Trigger Reset command
<i>Param4</i>	0	Reset

Returns

```
Struct InstrumentControlStruct
{
    int header;
    int errbyte;
    int detector;
    int cmdType;
    int value;
}
```

<i>header</i>	H_NO_ERROR	100
	H_INSTRUMENT_CONTROL_ERROR	900
<i>errbyte</i>	NO_ERROR	0
	PARAM_ERROR	-19
<i>detector</i>	2	Vnir
<i>cmdType</i>	4	Trigger Reset
<i>values</i>	0 -	Reset

Example

“IC,2,4,0” Resets the Trigger by turning off the LEDs and resetting the register.

INIT,0,x – Gets parameter from flash

Description:

This command gets a parameter stored in flash.

Note: This command requires a RESTORE command to have been called prior to retrieving the parameter values.

Parameters

<i>Param1</i>	"INIT"	Identifies the INIT command.
<i>Param2</i>	0	Gets a parameter from flash.
<i>Param3</i>	30 chars	Parameter name. See RESTORE command for possible names.
<i>Param4</i>	Not Used	

Returns

Struct ParamStruct

```
{
    int header;
    int errbyte;
    char name[30];
    double value;
    int count;
}
```

<i>header</i>	H_NO_ERROR	100
	H_INIT_ERROR	400
<i>errbyte</i>	NO_ERROR	0
	MISSING_PARAMETER	-8
<i>name</i>	Name of parameter up to 30 character long.	
<i>value</i>	Corresponding data value for parameter.	
<i>count</i>	The number of used entries.	

Example

"INIT,0,SerialNumber" Returns the Serial Number stored in Flash.

INIT,1,x,x – Adds a parameter to flash

Description:

This command adds a parameter to be stored in flash.

Note: This command requires the Save command to permanently store the value in flash.

Parameters

<i>Param1</i>		
	“INIT”	Identifies the INIT command.
<i>Param2</i>		
	1	Adds a parameter to flash.
<i>Param3</i>		
	30 chars	Parameter name
<i>Param4</i>		
	Double	Value of the Parameter

Returns

Struct ParamStruct

```
{
    int header;
    int errbyte;
    char name[30];
    double value;
    int count;
}
```

header

H_NO_ERROR	100
H_INIT_ERROR	400

errbyte

NO_ERROR	0
INI_FULL	-7

name

Name of parameter up to 30 character long.

value

Corresponding data value for parameter.

count

The number of used entries.

Example

“INIT,1,SerialNumber,4012” Adds the SerialNumber parameter with a value of 4012 to Flash.

INIT,2,x,x – Changes a parameter stored in flash

Description:

This command changes a parameter stored in flash.

Note: This command requires a RESTORE command to have been called prior to changing the parameter values. This command also requires the Save command to permanently store the value in flash.

Parameters

<i>Param1</i>		
	“INIT”	Identifies the INIT command.
<i>Param2</i>		
	2	Changes a parameter in flash.
<i>Param3</i>		
	30 chars	Parameter name. See RESTORE command for possible names
<i>Param4</i>		
	Double	Value of the Parameter

Returns

Struct ParamStruct

```
{
    int header;
    int errbyte;
    char name[30];
    double value;
    int count;
}
```

<i>header</i>		
	H_NO_ERROR	100
	H_INIT_ERROR	400
<i>errbyte</i>		
	NO_ERROR	0
	MISSING_PARAMETER	-8
<i>name</i>		
	Name of parameter up to 30 character long.	
<i>value</i>		
	Corresponding data value for parameter.	
<i>count</i>		
	The number of used entries.	

Example

“INIT,1,SerialNumber,6027” Changes the SerialNumber parameter to 6027 in Flash.

OPT,1 – Optimize VNIR detector

Description:

This command optimizes the VNIR detector.

Parameters

<i>Param1</i>	“OPT”	Identifies the OPT command.
<i>Param2</i>	1	VNIR detector (BITMASK = 0x01)
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct OptimizeStruct

```
{
    int header;
    int errbyte;
    int itime
    int gain[2]
    int offset[2]
}
```

<i>header</i>	H_NO_ERROR	100
	H_OPTIMIZE_ERROR	800
<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	MISSING_PARAMETER	-8
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	VNIR_OPT_ERROR	-15
	SWIR1_OPT_ERROR	-16
	SWIR2_OPT_ERROR	-17
	ABORT_ERROR	-18
<i>itime</i>	-1	Error if gain and offset are -1
	-1 - 15	Integration time for the VNIR detector.
<i>gain</i>	-1	Error
	[1] 0 – 4096	gain value for first SWIR detector.
	[2] 0 – 4096	gain value for second SWIR detector.
<i>offset</i>		



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

-1	Error
[1] 0 – 4096	offset value for first SWIR detector.
[2] 0 – 4096	offset value for second SWIR detector.

Example

“OPT,1” Optimize VNIR detector.

OPT,2 – Optimize SWIR1 detector

Description:

This command optimizes the SWIR1 detector.

Parameters

<i>Param1</i>	“OPT”	Identifies the OPT command.
<i>Param2</i>	2	SWIR1 detector (BITMASK = 0x02)
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct OptimizeStruct

```
{
    int header;
    int errbyte;
    int itime
    int gain[2]
    int offset[2]
}
```

<i>header</i>	H_NO_ERROR	100
	H_OPTIMIZE_ERROR	800
<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	MISSING_PARAMETER	-8
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	VNIR_OPT_ERROR	-15
	SWIR1_OPT_ERROR	-16
	SWIR2_OPT_ERROR	-17
	ABORT_ERROR	-18
<i>itime</i>	-1	Error if gain and offset are -1
	-1 - 15	Integration time for the VNIR detector.
<i>gain</i>	-1	Error
	[1] 0 – 4096	gain value for first SWIR detector.
	[2] 0 – 4096	gain value for second SWIR detector.
<i>offset</i>		



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

-1	Error
[1] 0 – 4096	offset value for first SWIR detector.
[2] 0 – 4096	offset value for second SWIR detector.

Example

“OPT,2” Optimize SWIR1 detector.

OPT,3 – Optimize VNIR and SWIR1 detectors

Description:

This command optimizes the VNIR and SWIR1 detectors.

Parameters

<i>Param1</i>	"OPT"	Identifies the OPT command.
<i>Param2</i>	3	VNIR and SWIR1 detector
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct OptimizeStruct

```
{
    int header;
    int errbyte;
    int itime
    int gain[2]
    int offset[2]
}
```

<i>header</i>	H_NO_ERROR	100
	H_OPTIMIZE_ERROR	800

<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	MISSING_PARAMETER	-8
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	VNIR_OPT_ERROR	-15
	SWIR1_OPT_ERROR	-16
	SWIR2_OPT_ERROR	-17
	ABORT_ERROR	-18

<i>itime</i>	-1	Error if gain and offset are -1
	-1 - 15	Integration time for the VNIR detector.

<i>gain</i>	-1	Error
	[1] 0 – 4096	gain value for first SWIR detector.
	[2] 0 – 4096	gain value for second SWIR detector.

offset



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

-1	Error
[1] 0 – 4096	offset value for first SWIR detector.
[2] 0 – 4096	offset value for second SWIR detector.

Example

“OPT,3” Optimize VNIR and SWIR1 detectors.

OPT,4 – Optimize SWIR2 detector

Description:

This command optimizes the SWIR2 detector.

Parameters

<i>Param1</i>	“OPT”	Identifies the OPT command.
<i>Param2</i>	4	SWIR2 detector (BITMASK=0x04)
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct OptimizeStruct

```
{
    int header;
    int errbyte;
    int itime
    int gain[2]
    int offset[2]
}
```

<i>header</i>	H_NO_ERROR	100
	H_OPTIMIZE_ERROR	800

<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	MISSING_PARAMETER	-8
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	VNIR_OPT_ERROR	-15
	SWIR1_OPT_ERROR	-16
	SWIR2_OPT_ERROR	-17
	ABORT_ERROR	-18

<i>itime</i>	-1	Error if gain and offset are -1
	-1 - 15	Integration time for the VNIR detector.

<i>gain</i>	-1	Error
	[1] 0 – 4096	gain value for first SWIR detector.
	[2] 0 – 4096	gain value for second SWIR detector.

offset



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

-1	Error
[1] 0 – 4096	offset value for first SWIR detector.
[2] 0 – 4096	offset value for second SWIR detector.

Example

“OPT,4” Optimize VNIR and SWIR1 detectors.

OPT,5 – Optimize VNIR and SWIR2 detectors

Description:

This command optimizes the VNIR and SWIR2 detectors.

Parameters

<i>Param1</i>	"OPT"	Identifies the OPT command.
<i>Param2</i>	5	VNIR and SWIR2 detector
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct OptimizeStruct

```
{
    int header;
    int errbyte;
    int itime
    int gain[2]
    int offset[2]
}
```

<i>header</i>	H_NO_ERROR	100
	H_OPTIMIZE_ERROR	800

<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	MISSING_PARAMETER	-8
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	VNIR_OPT_ERROR	-15
	SWIR1_OPT_ERROR	-16
	SWIR2_OPT_ERROR	-17
	ABORT_ERROR	-18

<i>itime</i>	-1	Error if gain and offset are -1
	-1 - 15	Integration time for the VNIR detector.

<i>gain</i>	-1	Error
	[1] 0 – 4096	gain value for first SWIR detector.
	[2] 0 – 4096	gain value for second SWIR detector.

offset



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

-1	Error
[1] 0 – 4096	offset value for first SWIR detector.
[2] 0 – 4096	offset value for second SWIR detector.

Example

“OPT,5” Optimize VNIR and SWIR2 detectors.

OPT,6 – Optimize SWIR1 and SWIR2 detectors

Description:

This command optimizes the SWIR1 and SWIR2 detectors.

Parameters

<i>Param1</i>	“OPT”	Identifies the OPT command.
<i>Param2</i>	6	SWIR1 and SWIR2 detector
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct OptimizeStruct

```
{
    int header;
    int errbyte;
    int itime
    int gain[2]
    int offset[2]
}
```

<i>header</i>	H_NO_ERROR	100
	H_OPTIMIZE_ERROR	800

<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	MISSING_PARAMETER	-8
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	VNIR_OPT_ERROR	-15
	SWIR1_OPT_ERROR	-16
	SWIR2_OPT_ERROR	-17
	ABORT_ERROR	-18

<i>itime</i>	-1	Error if gain and offset are -1
	-1 - 15	Integration time for the VNIR detector.

<i>gain</i>	-1	Error
	[1] 0 – 4096	gain value for first SWIR detector.
	[2] 0 – 4096	gain value for second SWIR detector.

offset



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

-1	Error
[1] 0 – 4096	offset value for first SWIR detector.
[2] 0 – 4096	offset value for second SWIR detector.

Example

“OPT,6” Optimize SWIR1 and SWIR2 detectors.

OPT,7 – Optimize VNIR, SWIR1 and SWIR2 detectors

Description:

This command optimizes the VNIR, SWIR1 and SWIR2 detectors.

Parameters

<i>Param1</i>	“OPT”	Identifies the OPT command.
<i>Param2</i>	7	VNIR, SWIR1 and SWIR2 detector
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct OptimizeStruct

```
{
    int header;
    int errbyte;
    int itime
    int gain[2]
    int offset[2]
}
```

<i>header</i>	H_NO_ERROR	100
	H_OPTIMIZE_ERROR	800

<i>errbyte</i>	NO_ERROR	0
	NOT_READY	-1
	MISSING_PARAMETER	-8
	VNIR_NOT_READY	-12
	SWIR1_NOT_READY	-13
	SWIR2_NOT_READY	-14
	VNIR_OPT_ERROR	-15
	SWIR1_OPT_ERROR	-16
	SWIR2_OPT_ERROR	-17
	ABORT_ERROR	-18

<i>itime</i>	-1	Error if gain and offset are -1
	-1 - 15	Integration time for the VNIR detector.

<i>gain</i>	-1	Error
	[1] 0 – 4096	gain value for first SWIR detector.
	[2] 0 – 4096	gain value for second SWIR detector.

offset



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

-1	Error
[1] 0 – 4096	offset value for first SWIR detector.
[2] 0 – 4096	offset value for second SWIR detector.

Example

“OPT,7” Optimize VNIR, SWIR1 and SWIR2 detectors.

RESTORE,x – Loads the flash into RAM

Description:

This command loads the values stored in flash into RAM. In version 1.5, this command takes upwards to 10 seconds to complete.

Note: “RESTORE,1” is required for 1.5 version and greater for Acquire (A) command to work properly.

Parameters

<i>Param1</i>	“RESTORE”	Identifies the RESTORE command.
<i>Param2</i>	0	Restores INI only
	1	Restores INI and build calibration Arrays.
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct InitStruct

```
{
    int header;
    int errbyte;
    char name[200][30];
    double value[200];
    int count;
    int verify;
}
```

header

H_NO_ERROR	100
H_INIT_ERROR	400

errbyte

NO_ERROR	0
INSTRUMENT_INI_LOAD_ERROR	-1
VNIR_INI_LOAD_ERROR	-2
SWIR1_INI_LOAD_ERROR	-3
SWIR2_INI_LOAD_ERROR	-4

name

Space for 200 entries with 30 character names.

INI entries below

Version
SerialNumber
CalibrationNumber
InstrumentType

Detectors
 StartingWavelength
 EndingWavelength
 InstrumentType
 InstrumentHours
 InstrumentMinutes
 ConnectionIdleTimeout
 ConnectionOverrideTimeout
 OptType
 OptimizationLogEnabled
 OptimizationTimeOutSeconds
 EnableTrigger
 MotorCurrentAdjustment
 MotorCurrentThreshold
 BoardAssemblyVersion
 VDetectorType
 VRealChannels
 VStartingWavelength
 VEndingWavelength
 VUseLinear
 VCalWavelengthStart
 VCalWavelengthStep
 VCalStartingWavelengthBlockV
 VCalWavelengthStepBlockV
 VDeltaStepBlockV
 VDeltaSquareStepBlockV
 VDriftChannelStart
 VDriftChannelCount
 VStartingIntegrationTimeIndex
 VMinIntegrationTimeIndex
 VMaxIntegrationTimeIndex
 VDarkCurrentCorrection
 VDarkSampleCount
 VInterpolate
 VVertex
 S1DetectorType
 S1RealChannels
 S1StartingWavelength
 S1EndingWavelength
 S1IndexChannel
 S1DarkStart
 S1DarkSize
 S1AdjustOffset
 S1CalStartingWavelengthBlockA
 S1CalWavelengthStepBlockA
 S1DeltaStepBlockA
 S1DeltaSquareStepBlockA
 S1CalStartingWavelengthBlockB
 S1CalWavelengthStepBlockB
 S1DeltaStepBlockB
 S1DeltaSquareStepBlockB
 S1Interpolate
 S1Vertex
 S2DetectorType

S2RealChannels
S2StartingWavelength
S2EndingWavelength
S2IndexChannel
S2DarkStart
S2DarkSize
S2AdjustOffset
S2CalStartingWavelengthBlockA
S2CalWavelengthStepBlockA
S2DeltaStepBlockA
S2DeltaSquareStepBlockA
S2CalStartingWavelengthBlockB
S2CalWavelengthStepBlockB
S2DeltaStepBlockB
S2DeltaSquareStepBlockB
S2Interpolate
S2Vertex

value Corresponding data value for 200 entries.
count The number of used entries.
verify The checksum value.

Example

“RESTORE,1” Loads the flash into RAM and builds calibration arrays.

SAVE – Saves the values in RAM to flash

Description:

This command saves the parameters in RAM to flash.

Parameters

Param1
“SAVE” Identifies the SAVE command.

Param2
Not Used.

Param3
Not Used.

Param4
Not Used.

Returns

Struct InitStruct

```
{
    int header;
    int errbyte;
    char name[200][30];
    double value[200];
    int count;
    int verify;
}
```

header

H_NO_ERROR	100
H_FLASH_ERROR	500

errbyte

NO_ERROR	0
----------	---

name
Space for 200 entries with 30 character names.

value
Corresponding data value for 200 entries.

count
The number of used entries.

verify
The checksum value.

Example

“SAVE” Saves the parameters in RAM to flash.

V – Version

Description:

This command returns the version of the firmware.

Parameters

<i>Param1</i>	“V”	Identifies the Version command.
<i>Param2</i>	Not Used.	
<i>Param3</i>	Not Used.	
<i>Param4</i>	Not Used.	

Returns

Struct ParamStruct

```
{
    int header;
    int errbyte;
    char name[30];
    double value;
    int type;
}
```

<i>header</i>	H_NO_ERROR	100
<i>errbyte</i>	NO_ERROR	0
<i>name</i>	Version of the firmware.	
<i>value</i>	Version value.	
<i>type</i>	Type of instrument	
	VNIR	1
	SWIR1	4
	VNIR/SWIR1	5
	SWIR2	8
	VNIR/SWIR2	9
	SWIR1/SWIR2	12
	VNIR/SWIR1/SWIR2	13

Example

“V” Returns the Version of the firmware.

Dark Current Collection

Dark Current collection is the process of blocking light coming into the instrument, then collecting the internal generated signal so that it can be subtracted from the external signal. Blocking the incoming light into the instrument can be accomplished with a mechanical shutter or by capping the fiber. A more efficient way of collecting dark current is through a dark current look up table. Recent testing has shown the dark current in the VNIR region to be stable. This stability allows for the use of a table to record the dark current values. The dark current table is easily generated with the Dark Current Calibration (DCC) utility supplied as part of the software package. Use of the table improves data collection rates by eliminating the time needed for the mechanical shutter process. Any changes in the dark current values due to normal fluctuations are small and are automatically adjusted by the software's Drift Lock feature. The use of the dark current table will be the default configuration on new instruments and can also be retroactively applied to existing Ethernet instruments.

The following is the Dark Correction algorithm:

$$\forall i \in \{0, \dots, n\} DC_S(i) = T_S(i) - D_S(i) + (V_{DarkCurrentCorrection} + (T_{drift} - D_{drift}))$$

Where:

n = size of the VNIR spectrum

DC_S = dark corrected spectrum

T_S = current measured spectrum

D_S = dark measured spectrum

$V_{DarkCurrentCorrection}$ = dark current correction constant

T_{drift} = current measured drift value

D_{drift} = dark measured drift value

The following describes the Dark Current Collection process for the three different methods:

- Has Shutter
- Has Dark File
- No Shutter or Dark File

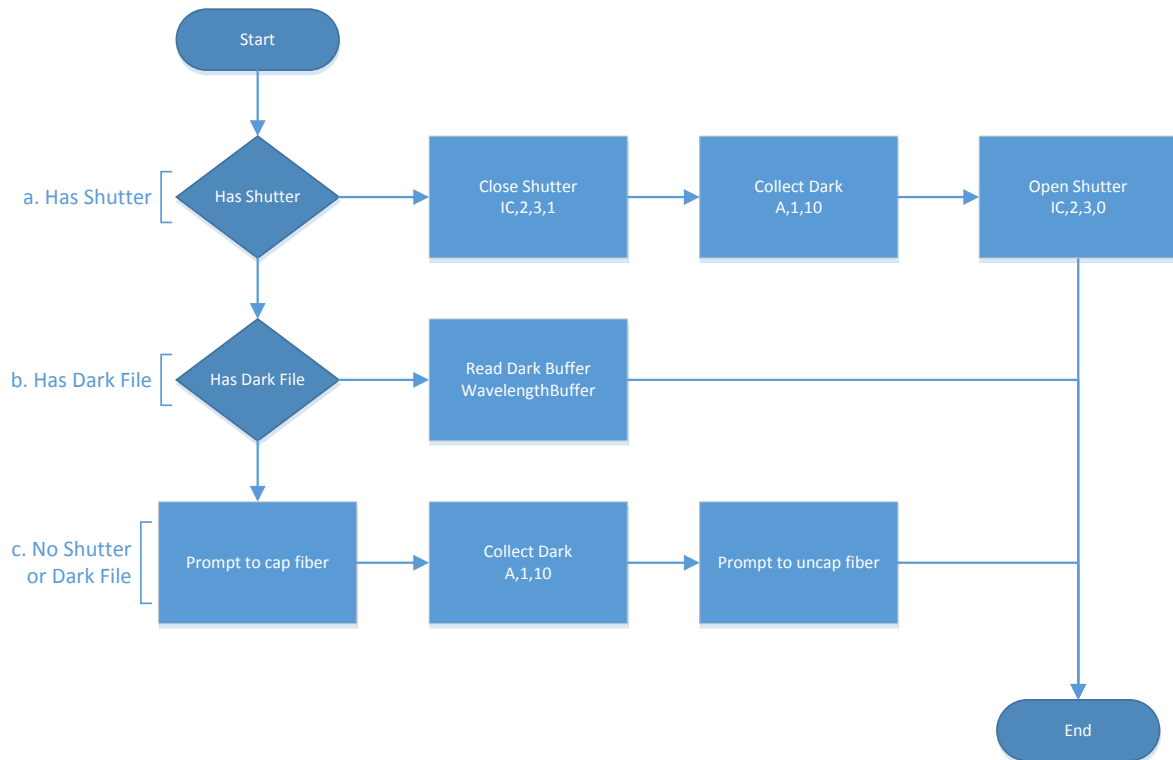


Figure 1: Dark Current Collection Process

1. Block Incoming Light
 - a. Has Shutter
Close Shutter – IC,2,3,1
 - b. Has Dark File
Open Dark Current ini file. This is will be in the form *<serial number>_<calibration number>_DarkCurrent.ini* (ie. 18343_2_DarkCurrent.ini). Where *<serial number>* is the serial number of the instrument and *<calibration number>* is the calibration number for the instrument.
 - c. No Shutter or Dark File
Prompt to cap the fiber.
2. Collect Dark Measured Spectrum - D_S
 - a. Has Shutter
Acquire spectrum from instrument – A,1,10
 - b. Has Dark File
Read the *WavelengthBuffer* from dark current file where the *Index* matches the current Integration Time. The look up table consists of channel data and wavelength data for each integration time.
 - c. No Shutter or Dark File
Acquire spectrum from instrument – A,1,10.
3. Read Dark Drift of Dark Measured Spectrum - D_{drift}
 - a. Has Shutter

- Read the *drift* value from Vnir Header.
 - b. Has Dark File
Read the *drift* value from dark current file where the Index matches the current Integration Time.
 - c. No Shutter or Dark File
Read the *drift* value from Vnir Header.
4. Collect Current Measured Spectrum - T_s
 - a. Has Shutter
Acquire spectrum from instrument – A,1,10.
 - b. Has Dark File
Acquire spectrum from instrument – A,1,10.
 - c. No Shutter or Dark File
Acquire spectrum from instrument – A,1,10.
5. Read Dark Drift of Current Measured Spectrum - T_{drift}
 - a. Has Shutter
Read the *drift* value from Vnir Header
 - b. Has Dark File
Read the *drift* value from dark current file where the Index matches the current Integration Time.
 - c. No Shutter or Dark File
Read the *drift* value from Vnir Header.
6. Compute Dark Corrected Spectrum - DC_s

Note: *VNIR DarkCurrentCorrection constant, VNIR StartingWavelength and EndingWavelength can be obtained from the Instrument using the INIT command.*

VNIR StartingWavelength

$V_{StartingWavelength} = INIT, 0, V_{StartingWavelength}$

VNIR EndingWavelength

$V_{EndingWavelength} = INIT, 0, V_{EndingWavelength}$

VNIR DarkCurrentCorrection constant

$V_{DarkCurrentCorrection} = INIT, 0, V_{DarkCurrentCorrection}$

Loop through the VNIR spectrum, subtract the dark spectrum from the current spectrum and add the Drift correction.

```
for(int i = 0; i < VEndingWavelength - VStartingWavelength; i++)
{
     $DC_s(i) = T_s(i) - D_s(i) + (V_{DarkCurrentCorrection} + (T_{drift} - D_{drift}))$ 
}
```


Writing a TCP Client

A TCP Client application is required to initiate a connection and issue commands to the TCP Server. A sample application has been provided to demonstrate the topics below. The sample application is located under the samples folder.

Making and closing a connection

To connect to a TCP Server, the TCP Client application must know the IP Address and Port number of the TCP Server. Please refer to the *Determine the network configuration* section for setting the TCP Server's IP Address. The ASD Instrument's IP address is 169.254.1.11. The Port number is 8080.

Connecting

The following code snippet shows how to make a connection to a TCP server with an address of 169.254.1.11 on port 8080.

```
//
// Initialize WSA
//
if(WSAStartup(MAKEWORD(2,2), &WsaDat)!=0)
{
    printf("WSA Initialization failed.");
    return;
}
//
// Create Socket
//
Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP );

if(Socket == INVALID_SOCKET)
{
    printf("Socket creation failed.");
}
//
// Connect to TCP Server
//
SOCKADDR_IN SockAddr;

SockAddr.sin_port = htons(8080);
SockAddr.sin_family = AF_INET;
SockAddr.sin_addr.S_un.S_addr = inet_addr("169.254.1.11");

int RetVal = connect(Socket, (SOCKADDR *)&SockAddr, sizeof(SockAddr));
if(RetVal != 0)
{
    int l = WSAGetLastError();
    printf("Failed to establish connection with server. %d\n", l);
}
```

Closing the Connection

```
//  
// Close the Socket  
//  
closesocket(Socket);  
  
//  
// Clean of the Winsock library  
//  
WSACleanup();
```

The following code snippet shows how to disconnect from the TCP Server.

Reading the starting and ending wavelength

Before reading the starting and ending wavelength of the TCP Server, the instrument's INI must be loaded into flash. Each instrument comes with the INI pre loaded. To update the instrument's INI, please refer to the Net Configuration Guide. Reading the instrument's starting and ending wavelength uses the INIT,0,x command. The following code snippet demonstrates reading the starting and ending wavelength.

Starting Wavelength

```
CString strCommand = "INIT,0,StartingWavelength");  
  
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );
```

Ending Wavelength

```
CString strCommand = "INIT,0,EndingWavelength");  
  
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );
```

Optimize

The following code snippet demonstrates how to optimize the instrument.

```
CString strCommand = "OPT,7";  
  
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );
```

Acquiring data

The following code snippet demonstrates how to Acquire data from the instrument.

```
//  
// Initialize the FR Spectrum Structure  
//
```

```
FRInterpSpecStruct *iss;

iss = (FRInterpSpecStruct *)malloc(sizeof(*iss));
//
// Collect 10 samples
//
CString strCommand = "A,1,10";

bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );

//
// Loop until the data has been collected
//
int bytesRecv = 0;
char *recvbuf = new char[bytesToRecv];
totalBytesRecv = 0;

while( totalBytesRecv < bytesToRecv )
{
    bytesRecv = recv( Socket, recvbuf, bytesToRecv, 0 );
    if (bytesRecv == SOCKET_ERROR)
        break;

    if ( bytesRecv == 0 || bytesRecv == WSAECONNRESET )
    {
        printf( "Connection Closed.\n");
        break;
    }
    printf( "Bytes Recv: %ld\n", bytesRecv );

    memmove(&recvBuf[totalBytesRecv], recvbuf, bytesRecv);
    totalBytesRecv += bytesRecv;
}

//
// Convert the Header and errbyte from big endian to little endian to see if it is good data
//
iss->FRHeader.Header = ntohl(iss->FRHeader.Header);
iss->FRHeader.errbyte = ntohl(iss->FRHeader.errbyte);

if(iss->FRHeader.Header == 100)
{
    unsigned long z;
    //
    // Convert the buffer from big endian to little endian and store the value as a float
    //
    for(int i=0;i<(sizeof(iss->SpecBuffer) / sizeof(float));i++)
    {
        z = ntohl(iss->SpecBuffer[i].i);
        memcpy(&iss->SpecBuffer[i].f,&z,sizeof(float));
    }
}
```

Displaying a Dark Corrected Spectrum

The following code snippet demonstrates how to display a dark corrected spectrum using a shutter.

```
//  
// Close the shutter  
//  
CString strCommand = "IC,2,3,1");  
  
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );  
//  
// Initialize the FR Dark Spectrum Structure  
//  
FRInterpSpecStruct *issDarkSpectrum;  
  
issDarkSpectrum = (FRInterpSpecStruct *)malloc(sizeof(*issDarkSpectrum));  
//  
// Collect 10 Dark Samples  
//  
CString strCommand = "A,1,10";  
  
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );  
  
//  
// Convert the received data to float  
//  
..... Code omitted for brevity – See Acquire section for details  
//  
// Assign Dark drift value  
dark_drift = issDarkSpectrum.FRHeader.v_header.drift;  
//  
// Open the shutter  
//  
strCommand = "IC,2,3,0");  
  
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );  
//  
// Initialize the FR Spectrum Structure  
//  
FRInterpSpecStruct *iss;  
  
iss = (FRInterpSpecStruct *)malloc(sizeof(*iss));  
  
//  
// Acquire data to subtract the dark  
//  
strCommand = "A,1,10";  
  
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );  
  
//  
// Convert the received data to float  
//  
..... Code omitted for brevity – See Acquire section for details
```

```
//
// Assign Current drift value
current_drift = iss.FRHeader.v_header.drift;

//
// Subtract the Dark Spectrum from the current spectrum
//
if(iss->FRHeader.Header == 100)
{
    // Compute drift
    float drift = m_iVnirDarkCurrentCorrection + (current_drift - dark_drift);
    // Subtract dark
    for(int i = 0; i < ((m_iVnirEndingWavelength + 1) - m_iStartingWavelength); i++)
        iss->SpecBuffer[i].f -= issDarkSpectrum->SpecBuffer[i].f + drift;
}
```

Displaying a Reflectance Spectrum

The following code snippet demonstrates how to display a reflectance spectrum.

```
//
// Collect and store a reference spectrum
//

//
// Initialize the Reference FR Spectrum Structure
//
FRInterpSpecStruct *issReference;

issReference = (FRInterpSpecStruct *)malloc(sizeof(*issReference));

CString strCommand = "A,1,10";

bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );

//
// Convert the received data to float
//
..... Code omitted for brevity – See Acquire section for details
//
//
// Collect a current Spectrum to compute reflectance
//
//
// Initialize the FR Spectrum Structure
//
FRInterpSpecStruct *iss;

iss = (FRInterpSpecStruct *)malloc(sizeof(*iss));

//
// Acquire current data
//
strCommand = "A,1,10";
```

```
bytesSent = send( Socket, strCommand, strCommand.GetLength(), 0 );

//
// Convert the received data to float
//
..... Code omitted for brevity – See Acquire section for details
//
//
// Compute reflectance
//
if(iss->FRHeader.Header == 100)

{

    // Compute Reflectance

    for(int i = 0; i < ((m_iEndingWavelength + 1) - m_iStartingWavelength); i++)

        iss->SpecBuffer[i].f = iss->SpecBuffer[i].f / issReference->SpecBuffer[i].f;

}
```

Normalizing a Spectrum

The following code snippet demonstrates how to normalize spectrum.

```
//
// Acquire data – see the Acquire section
//
// Create the Normalized structure
//
FRInterpSpecStruct *issNormalize;
issNormalize = (FRInterpSpecStruct*)malloc(sizeof(*issNormalize));

if(iss->Header == 100)
{

    int i;
    // Normalize Vnir to IT-17ms
    for(i = 0; i < ((m_iVnirEndingWavelength + 1) - m_iStartingWavelength); i++)
        issNormalize->SpecBuffer[i].f = iss->SpecBuffer[i].f / (1<<it);

    // Normalize Swir1 Gain to 4096
    float gc = 256;
    float n = slg/gc;
    for(i = (m_iVnirEndingWavelength + 1) - m_iStartingWavelength;
        i < ((m_iSwir1EndingWavelength + 1) - m_iStartingWavelength); i++)
        issNormalize->SpecBuffer[i].f = iss->SpecBuffer[i].f * n;

}
```



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

```
// Normalize Swir2 Gain to 4096
n = s2g/gc;
for(i = (m_iSwir1EndingWavelength + 1) - m_iStartingWavelength;
    i < ((m_iSwir2EndingWavelength + 1) - m_iStartingWavelength); i++)
    issNormalize->SpecBuffer[i].f = iss->SpecBuffer[i].f * n ;

}
```



ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: (303) 444-6522
Fax: (303) 444-6825
Email: nir.support@panalytical.com

Support

ASD Inc., a PANalytical company
2555 55th Street, Suite 100
Boulder, CO 80301

Phone: 303-444-6522
Fax: 303-444-6825
Web site: www.asdi.com
Email: nir.support@panalytical.com