# FTLight File/Stream Datenstruktur und Datenaustausch

## Version

2023-06-18 PDF: http://wegalink.eu/development/ftlight/FTLight_DE.pdf

### Entwicklungsverlauf

2023-06-18 Identifikator-Datentyp FTLI auf jenseits des Universums erweitert
2022-05-15 Algorithmische Antwort zum Realisieren aktiver Elemente eingeführt
2022-01-18 verschränkte BinDIF Messwerte bei geringer Schwankungsbreite eingeführt
2021-02-05 Synchronschreiben weiter erläutert und detaillierter beschrieben
2021-01-25 Darstellung von Zeitangaben und physikalischen Werten vervollständigt
2021-01-20 Zeitangaben vor 1970-01-01 00:00:00 UTC als negative Zeiten ergänzt
2020-12-13 Zeitsynchronisation, Zeitstempel bei Anfragen als verbindlich festgelegt
2020-09-24 Format BinDIF für differenzielle Kodierung von Datenströmen ergänzt
2018-03-24 Kodierung physikalischer Einheiten und Kombination von Formaten ergänzt
2018-01-07 Arrays von Zahlen/Text mit beliebiger Anzahl von Dimensionen ergänzt
2016-05-21 Abonnieren von aktuellen Daten bei Änderungen (Ereignissen)
2015-09-26 Neuer Arbeitstitel „FTLight“ (Faster than LIGHT, Schneller als LICHT)
2015-08-08 Rahmen für Entropie-Modus mit Datenraten bis nahezu 100% der Bandbreite
2015-05-16 Entropie-Modus für hohe Datenraten bis zu 87% der Bandbreite, z.B FPGA
2015-04-11 Fortsetzen eines vorherigen Synchronschreibens ergänzt
2014-11-22 Prüfsumme am Zeilenende optional festgelegt
2014-11-18 Mehrdeutigkeit für Datentyp am Zeilenanfang aufgelöst
2014-11-16 Daten Update/Speichern geändert (Danke an unsere Partner für die Hinweise)
2014-11-02 Erläuterungen zum Konzept ergänzt, Beispiele konsolidiert
2014-11-01 Identifikator auf universelle Eindeutigkeit erweitert
2014-10-31 Cosmos CmXtoken, CmXlink ergänzt, an neue Rechtschreibung angepasst
2005-04-16 Entwerten von Sonderzeichen auf Backslash geändert
2005-03-20 Datentyp für Wertedarstellung ergänzt
2005-03-16 Framework-Funktionalität ergänzt
2005-03-09 Zeitdatentyp ergänzt
2005-03-06 Datentypen, Kapselung, Arrays und schnelle Binärsignalkodierung ergänzt
2005-03-03 Zahlendarstellung im Binärformat ergänzt
2005-02-26 Beispiele für Metadaten und Mehrkanalempfänger ergänzt
2005-02-20 Detaillierte Festlegungen zu Zahlenformaten
2005-02-17 Hexadezimalzahlen als Format ergänzt, Punkt im Identifikator
2004-03-10 Einführung mit allgemeinem Überblick zum Zweck ergänzt

| | |
|---|---|
| 2004-01-29 | Versionsverwaltung ergänzt |
| 2004-01-22 | Request/Response durch Datenflusselemente ergänzt |
| 2004-01-21 | Request/Response-Fähigkeiten ergänzt |
| 2004-01-20 | FTLight-Speicherorganisation ergänzt |
| 2004-01-19 | Unterstützung für Datenintegrität ergänzt |
| 2004-01-18 | Erklärung für ‘FTLight-Collection’ ergänzt und Adressverwendung erweitert |
| 2004-01-12 | Erste Version basierend auf Diskussionen in der ERAC-VLBI-Gruppe |
| 1997-09 | Anforderungen für Datenaustausch vom 1. ERAC-Kongress in Heppenheim |

# Mitarbeit

Folgende Personen haben an der Diskussion über den Inhalt dieses Dokumentes teilgenommen und viele Ideen eingebracht, welche im Dokument eingearbeitet sind:


| | | |
|---|---|---|
| Armin Falb | falbs@t-online.de | |
| Marko Cebokli | marko.cebokli@mors.si | |
| Jim Sky | radiosky@radiosky.com | http://www.radiosky.com |

# Disclaimer

```
////////////////////////////////////////////////////////////////////////////
//
// FTLight.pdf: Dokumentation der FTLight File/Stream-Datenstruktur
//
////////////////////////////////////////////////////////////////////////////
//
// Autor:      Eckhard Kantz
// eMail:      software@wegalink.eu
// Motivation: European Radio Astronomy Club (ERAC),
//             Projekt ALLBIN (http://eracnet.org/workshop/allbin.htm)
// Software:   Prototyp-Implementierung auf GitHub unter dem Vorbehalt einer
//             nicht vollständigen Umsetzung der Spezifikation:
//             https://github.com/WegaLink/FTLight
//
////////////////////////////////////////////////////////////////////////////
```

# Inhaltsverzeichnis

# Zielstellung

Die FTLight File/Stream-Datenstruktur wurde mit dem Ziel entworfen, die Sammlung und den Austausch großer Datenmengen zu unterstützen sowie den Zugriff auf einzelne Bestandteile bis hinunter zu den kleinsten Detailinformationen zu ermöglichen.

Ein spezieller Bedarf für den Umgang mit großen Datenvolumen besteht insbesondere im Bereich Radioastronomie, zum Beispiel wenn die Datenströme von verteilten Antennenstandorten zusammengeführt werden sollen, um alle Anlagen zu einem integrierten Antennenkomplex zu verbinden, so wie dies für das Generieren von Himmelsansichten mit Interferometer-Netzwerken der Fall ist.

Obwohl diese Spezifikation auf die besonderen Anforderungen von Interferometer-Netzwerken ausgelegt ist, so ist sie darüber hinaus auch für andere Zusammenschaltungen von Computern anwendbar, um diese in integrierter Weise zusammenarbeiten zu lassen.

## Einführung

In der gegenwärtigen Computermethodologie ist das Schichtenmodell weit verbreitet, um Computersysteme miteinander zu verbinden. Schichtenmodell bedeutet das Umhüllen von zu transportierenden Informationspaketen mit Zusatzinformationen, zum Beispiel wo die Information herkommt, wo sie hingeht, welchen Kategorien sie angehört und anderen. Das so entstehende Datenpaket ist möglicherweise in ein übergeordnetes Datenpaket eingebettet, welches den gleichen Zweck verfolgt, lediglich auf einer höheren Ebene. Das entstehende Datenpaket kann wiederum von einem nächsten Datenpaket umhüllt sein, und so weiter.

In der Regel werden notwendige Steuerinformationen im Vorspann (Header) eines Datenpaketes abgelegt, während die zu transportierende Information im Hauptteil des Datenpaketes (body) eingefügt wird. Weiterhin können in einem Nachsatz (Trailer) weitere Informationen enthalten sein, welche zum Beispiel zur Sicherstellung der Datenintegrität von Bedeutung sind. Die Struktur des endgültigen Datenpaketes, welches effektiv über den Verbindungskanal übertragen wird, kann somit sehr komplex werden und der Overhead kann dementsprechend groß sein:

Vorspann1 Vorspann2 ... Vorspann N (effektive Information) Nachsatz N ... Nachsatz2 Nachsatz1

Ein Vorteil des Schichtenmodells ist offensichtlich, dass die einzelnen Schichten unabhängig voneinander sind und dass sie separat implementiert werden können. Weiterhin können geeignete Kombinationen von Transportschichten benutzt werden, um Informationen zwischen beliebigen heterogenen Systemen auszutauschen.

Andererseits erhöht jede Transportschicht den Umfang der zu übertragenden Information und die Forderung nach unabhängigen Schichten macht es nahezu unmöglich, Transportoptimierungen über mehrere Schichten hinweg durchzuführen. In einem extremen Fall ergab die Analyse eines Datenstromes, welcher Tabellen-artige Daten mit mehreren tausend Einträgen transportierte, dass der resultierende Overhead bei Anwendung des SOAP-Protokolls 95% betrug und nur 5% Nutzdaten im Vergleich zur binären Repräsentation der gleichen Daten übertragen wurden.

Es sei erwähnt, dass sich das SOAP-Protokoll sehr gut für die Übertragung beliebig strukturierter Daten eignet, was bei anderen Protokollen oft nicht der Fall ist. Unter existierenden Protokollen musste daher in der Regel für einen konkreten Einsatzfall nach einem Kompromiss zwischen Effektivität der Übertragung und Möglichkeiten strukturierter Datenrepräsentation gesucht werden.

Die FTLight-Spezifikation verfolgt das Ziel, das Beste aus beiden Welten miteinander zu verbinden, also zum einen bei der Speicherung und Übertragung hoch effektiv zu sein und andererseits beliebig strukturierte Daten abbilden zu können. Hinzu kommen weitere Eigenschaften im Zusammenhang mit der Notwendigkeit des Umgangs mit beliebig großen Datenvolumen.

Ein weiteres Ziel besteht in der Unabhängigkeit von jeglichen Transportschichten, welche den Informationsfluss zwischen Computern steuern. Für einen erfolgreichen Datenaustausch sollte es genügen, eine Verbindung auf der Basis von IP-Services zu haben, welche das Spezifizieren des Empfängers der Information ermöglichen. Alternativ ist es möglich, auch jede andere Kopplung, wie zum Beispiel serielle Verbindungen für einen FTLight-basierten Informationsaustausch zu verwenden. Die Implementierung dieser Zielstellung wird eine breite Anwendung in heterogenen Umgebungen ermöglichen und gleichzeitig stellt dies eine gute Basis für sorgfältige und tiefgehende Transportoptimierungen dar.

Spezielle Aufmerksamkeit muss der Gesamtsignallaufzeit gewidmet werden die entsteht, wenn ein System ein Datenpaket an ein zweites System sendet und wenn dieses eine Antwort an das erste System zurücksendet. Je mehr Signalläufe für das reguläre Übertragen von kurzen Informationen erforderlich sind, desto mehr Bedeutung kommt dieser Frage zu. Im Falle von extremen Signallaufzeiten, wie sie zum Beispiel bei der Kommunikation von Weltraumsonden mit Bodenstationen auftreten, steht maximal ein kompletter Signallauf für das Übertragen von Informationen zur Verfügung. Alle Datenprotokolle, welche mehrere Signalläufe benötigen, sind in diesem Fall nutzlos und scheiden für die Anwendung aus.

Kurz zusammen gefasst, besteht das Ziel dieser Spezifikation in einer eindeutigen selbsterklärenden hierarchischen Datenstruktur, welche mit höchster Effektivität zwischen einem Sender und einem Empfänger übertragen werden kann.

# Fallstudien

## A – Datenreihe

Eine der einfachsten Datenstrukturen wird durch eine einzelne Messwertspalte dargestellt:

```
2602
2595
2594
..
```

Der wesentliche Nachteil einer einzelnen Messwertspalte ist das Fehlen jeglicher Informationen über Herkunft und Bedeutung der Daten sowie die fehlende Maßeinheit für die Zahlen.

## B – Datensatz mit Zeitstempel

Ein Vorspann (Header) sowie Zeitstempel liefern bereits mehr Informationen über die Messwerte:

```
Zeit                          Flux        Temperatur
[Sekunden seit 1.1.1970]      [Jy]        [°C]
1073217600.370                2602        -2.4
1073217600.390                2595        -2.4
1073217600.410                2594        -2.3
```

Mit Header-Informationen und Zeitstempeln wird die Bedeutung der Messwerte bereits deutlicher. Jedoch fehlen auch in diesem Fall Informationen über die beobachtete Radioquelle sowie Informationen zur verwendeten Messausrüstung.

## C – Strukturierte Information

Eine Informationsdarstellung in strukturierter Weise klärt alle Eigenschaften der Messwertdaten bis hin zu den kleinsten Details:

```
EKD@JO63rx_Dambeck.RSpectro,1073217600
,Antenne,Parabolspiegel 90cm
,Azimut:Grad,0
,Elevation:Grad,15
,Frequenz:GHz,10.600
,Bandbreite:kHz,250
0:Zeit,Flux,Temperatur
[Sekunden seit 1.1.1970],[Jy],[°C],@
1073217600.370,2602,-2.4,1073217600.590
1073217600.390,2595,-2.4,1073217600.615
1073217600.410,2594,-2.3,1073217600.640
```

Der obenstehende strukturierte Informationsblock erweitert die vorherige Messwertdarstellung mit Zeitstempeln um beschreibende Zusatzinformationen (Metainformationen), welche zum Beispiel Auskunft über den Entstehungszeitpunkt des Datensatzes (Erfassungszeit / 1.Spalte, Speicherzeit / letzte Spalte) sowie die benutzte Spezifikation geben. Weiterhin werden der Operator, der Messort sowie die verwendete Messausrüstung mittels Schlüssel (EKD@JO63rx_Dambeck.RSpectro) angegeben, ebenso die Beobachtungsdetails wie Antenne, Azimut, Elevation, Frequenz, Bandbreite.

## Bildung von strukturierten Informationen

Ein gebräuchlicher Weg für das Erzeugen von Strukturen bei einzelnen Informationen ist das Bilden von geordneten Informationsmengen, wodurch jeder einzelnen Information im Prinzip eine Reihenfolgenummer zugewiesen wird:

```
[0:1073217600, 1:Antenne, 2:Azimut, 3:Elevation, 4:Frequenz, 5:Bandbreite, 6:Zeit,
7:Flux, 8: Temperatur]
```

Anschließend können alle Einzelinformationen sowohl durch ihre Reihenfolgenummer als auch durch den von ihnen repräsentierten Wert referenziert werden, zum Beispiel Nummer "**1:**" ist gleichbedeutend mit "**Antenne**".

Grundsätzlich gibt es für die soeben beschriebenen Informationsmengen zwei Möglichkeiten zu deren Erweiterung, zum einen in der Größe und zum anderen in der Tiefe. Eine Erweiterung der Größe bedeutet, dass der Informationsmenge weitere Elemente hinzugefügt werden. Eine Erweiterung der Strukturtiefe hingegen bedeutet, dass neue Informationsmengen gebildet und diese einem Element der bestehenden Menge zugeordnet werden.

Beispiel:
```
[0: 1073217600, .. 4:Frequenz, ..]
                                |
                                |_____ [0:GHz, 1:10.600]
```

Die Erweiterung von Größe und Tiefe geordneter Informationsmengen führt zu hierarchischen Datenstrukturen. Das Adressieren eines Elementes innerhalb der hierarchischen Datenstruktur erfordert das Aufzählen aller übergeordneten Bestandteile dieser Struktur, welche somit den Pfad von einem obersten Element bis zum adressierten Element darstellen. Dabei können die übergeordneten Elemente sowohl mittels ihrer Werte als auch mit ihren Ordnungsnummern, welche ihnen in den jeweiligen geordneten Datenmengen zugeteilt wurden, referenziert werden.

Untenstehend ist ein Vergleich aufgelistet, welcher den Pfad aller Elemente der zuvor aufgeführten Beispieldatenstruktur einmal als Folge von Ordnungsnummern auf der linken Seite und zum anderen die Werte aller Elemente auf der rechten Seite zeigt:

| | |
|---|---|
| **0** | **EKD@JO63rx_Dambeck.RSpectro** |
| **0-0** | **1073217600** |
| **0-1** | **Antenne** |
| 0-1-0 | Parabolspiegel 90cm |
| **0-2** | **Azimut** |
| 0-2-0 | Grad |
| 0-2-1 | 0 |
| **0-3** | **Elevation** |
| 0-3-0 | Grad |
| 0-3-1 | 15 |
| **0-4** | **Frequenz** |
| 0-4-0 | GHz |
| 0-4-1 | 10.600 |
| **0-5** | **Bandbreite** |
| 0-5-0 | kHz |
| 0-5-1 | 250 |
| ***0-6*** | ***Zeit*** |
| 0-6-0 | [Sekunden seit 1.1.1970] |
| 0-6-0-0 | 1073217600.370 |
| 0-6-0-1 | 1073217600.390 |
| 0-6-0-2 | 1073217600.410 |
| ***0-7*** | ***Flux*** |
| 0-7-0 | [Jy] |
| 0-7-0-0 | 2602 |
| 0-7-0-1 | 2595 |
| 0-7-0-2 | 2594 |
| ***0-8*** | ***Temperatur*** |
| 0-8-0 | [°C] |
| 0-8-0-0 | -2.4 |
| 0-8-0-1 | -2.4 |
| 0-8-0-2 | -2.3 |

# Abbildung strukturierter Informationen in einem File/Stream

## Entwurfsziele

Unter verschiedenen Möglichkeiten zur Abbildung von strukturierten Informationen in einem File/Stream hat eine solche Lösung den Vorrang, welche bei vorgegebener Informationsmenge das geringste Datenvolumen für das File/ den Stream ergibt. Daher werden explizite Strukturelemente wie zum Beispiel Tags in XML-Dateien durch implizite Regeln ersetzt werden, welche in der überwiegenden Mehrzahl der Fälle ohne explizite Strukturelemente auskommen. In den verbleibenden Fällen wird die Strukturinformation in der zuvor dargestellten kurzen Form als Folge von Reihenfolgenummern hinzugefügt.

Zusätzlich zu den bereits aufgeführten Anforderungen besteht ein weiteres Entwurfsziel darin, dass die resultierenden FTLight Files/Streams lesbar dargestellt und einzelne Informationselemente mittels Texteditor angepasst werden können.

## File/Stream-Struktur

Ein File/Stream wird aus 8-Bit-Werten (Bytes) erzeugt, welche den Wertevorrat von 0 bis 255 darstellen. Einige ASCII-Zeichen, wie zum Beispiel 13 (CR-Wagenrücklauf) und 10 (LF-Zeilenschaltung) werden zum Strukturieren der Dokumente verwendet, während die Mehrzahl der Zeichen für die Darstellung von Informationen verwendet wird.

### Zeilen innerhalb eines Files/Streams

Die oberste Strukturebene einer File/Stream-Struktur (Zeile) wird durch eine Zeilenschaltung (CR+LF) erzeugt. Innerhalb einer Zeile sorgen spezielle Trennnzeichen für die Einteilung in einzelne Informationselemente.

### Informationselemente innerhalb einer Zeile

Für die Einteilung einer Zeile in Informationselemente finden vier verschiedene Trennzeichen Verwendung:

| | |
|---|---|
| 44 (Komma) | - Erweiterung von Pfad oder Informationsmenge, nächstes Informationselement ist Text oder numerisch |
| 59 (Semikolon) | - Erweiterung von Pfad oder Informationsmenge, nächstes Informationselement ist binär oder speziell |
| 58 (Doppelpunkt) | - Neubeginn einer Informationsmenge (auf einem Pfad), nächstes Informationselement ist Text oder numerisch |
| 61 (Gleichheitszeichen) | - Neubeginn einer Informationsmenge (auf einem Pfad), nächstes Informationselement ist binär oder speziell |

Die Erweiterungselemente (Komma, Semikolon) trennen, beginnend am Zeilenanfang, zunächst Pfadbestandteile voneinander, solange bis in der Zeile ein Startelement für den Neubeginn einer Informationsmenge (Doppelpunkt, Gleichheitszeichen) auftritt. Ab diesem Startelement werden von den gleichen Erweiterungselementen (Komma, Semikolon) Informationselemente auf einer Ebene der Hierarchie getrennt. Dies entspricht einer Aufzählung von Elementen auf dem aktuellen Pfad.

Beispiel:

```
Frequenz:GHz,10.600
```

entspricht folgender Struktur:

```
0               Frequenz
```

```
0-0                 GHz
0-1                 10.600
```

**Achtung:** Die Doppelbedeutung der Erweiterungselemente muss bei der Interpretation von FTLight Files/Streams beachtet werden, um die Struktur der Informationen korrekt zu rekonstruieren. Eine Reduktion der Strukturelemente durch Doppelbelegung stellt keine Einschränkung bei der Bildung von Informationsstrukturen dar. Sie ermöglicht jedoch das Auslassen der nicht druckbaren Zeichen 0..31 sowie das Reservieren von 216 Zeichen für das Kodieren von Binärinformationen (BinX).

### Duplizierte Informationselemente in aufeinanderfolgenden Zeilen

Sobald ein Informationselement an gleicher Stelle in einer nachfolgenden Zeile auftritt so wird dieses einfach weggelassen:

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,1073217600
EKD@JO63rx_Dambeck.RSpectro,Antenne,Parabolspiegel 90cm
```

Ist gleichbedeutend mit:

```
EKD@JO63rx_Dambeck.RSpectro,1073217600
,Antenne,Parabolspiegel 90cm
```

und entspricht in beiden Fällen der folgenden Datenstruktur:

```
0             EKD@JO63rx_Dambeck.RSpectro
0-0               1073217600
0-1               Antenne
0-1-0               Parabolspiegel 90cm
```

### Vergrößerung einer Informationsmenge durch eine Folgezeile

Das erste unterschiedliche Pfadelement in einer Folgezeile erweitert die Informationsmenge auf der entsprechenden Ebene:

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,1073217600
,Antenne,Parabolspiegel 90cm
```

Die erste Zeile gibt folgenden Pfad vor:

```
0             EKD@JO63rx_Dambeck.RSpectro
0-0               1073217600
```

während die zweite Zeile die Informationsmenge mit dem Element "**1073217600**" um ein neues Element "**Antenne**" erweitert:

```
0-1               Antenne
0-1-0               Parabolspiegel 90cm
```

### Verwalten eines aktuellen Pfades

Wenn eine Zeile einen Pfad spezifiziert so wird dieser zum aktuellen Pfad. Falls nachfolgende Zeilen ohne explizite Formatkennzeichnung gleich mit Text oder einem binären oder numerischen Element beginnen dann bleibt der vorherige Pfad erhalten und kommt als aktueller Pfad zur Anwendung.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro:Zeit,Flux,Temperatur
```

Ist gleichbedeutend mit:

```
EKD@JO63rx_Dambeck.RSpectro
Zeit,Flux,Temperatur
```

und auch mit:

```
EKD@JO63rx_Dambeck.RSpectro
0:Zeit,Flux,Temperatur
```

Und entspricht in allen Fällen der folgenden Datenstruktur:

```
0            EKD@JO63rx_Dambeck.RSpectro
0-0             Zeit
0-1             Flux
0-2             Temperatur
```

Das "**EKD@JO63rx_Dambeck.RSpectro**"-Element wird als Pfadursprung (Root) erkannt da es ein ‚@'-Zeichen enthält, so wie es bei „Datentypen" definiert wird.

### Verwalten einer übergeordneten (Parent-)Informationsmenge

Die auf dem aktuellen Pfad neu erzeugten Elemente einer Informationsmenge werden zur übergeordneten (Parent-) Informationsmenge für nachfolgende synchrone Schreiboperationen.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro
Zeit,Flux,Temperatur
```

Die neu erzeugten Elemente [Zeit,Flux,Temperatur] der Informationsmenge werden zur übergeordneten Informationsmenge auf dem aktuellen Pfad [`EKD@JO63rx_Dambeck.RSpectro`].

### Synchrone Schreiboperationen

Oft bestehen Datensammlungen aus mehreren Spalten. Das synchrone Schreiben unterstützt das Hinzufügen eines weiteren Datensatzes in solch einer Datensammlung mittels einer Zeile. Die Elemente dieser Zeile erweitern die entsprechenden Informationsmengen, welche den Elementen der übergeordneten (Parent-)Informationsmenge als untergeordnete (Child-)Informationsmengen zugeordnet sind, solange nachfolgende Zeilen ohne explizite Formatkennzeichnung gleich mit Text oder einem binären oder numerischen Element beginnen. Dies entspricht dem Hinzufügen einer weiteren, untergeordneten Überschriftenzeile in einer Tabelle.

Sehr häufig erfolgen bei Tabellendaten anschließend mehrere synchrone Schreiboperationen ohne Wechsel der übergeordneten (Parent-) Informationsmenge (der Header- oder Überschriftenzeile). Falls eine Informationsmenge zu so einer neuen feststehenden (Parent-) Informationsmenge werden soll, dann wird der Zeile, welche diese Elemente enthält, ein einzelnes ‚@'-Zeichen als letztes Element der Zeile nachgestellt. Unterhalb des '@'-Zeichens (in der Child-Informationsmenge) werden optional die Systemzeit des Speicherns im gleichen Format wie der Zeitstempel am Zeilenanfang und zusätzlich optional die Reihenfolgenummer jedes Datensatzes beginnend mit 1 als nächstes Element nach dem Zeitstempel eingetragen.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro
Zeit,Flux,Temperatur
[Sekunden seit 1.1.1970],[Jy],[°C],@
```

Die Elemente der dritten Zeile werden den Informationsmengen zugeordnet, welche mit den Elementen der zweiten Zeile verknüpft sind während sie gleichzeitig die Rolle der übergeordneten (Parent-) Informationsmenge für nachfolgende Zeilen übernehmen. Die Datenstruktur sieht daher folgendermaßen aus:

```
0          EKD@JO63rx_Dambeck.RSpectro
0-0           Zeit
0-0-0           [Sekunden seit 1.1.1970]
0-1           Flux
```

```
0-1-0         [Jy]
0-2          Temperatur
0-2-0         [°C]
0-3          <leer>
0-3-0         @
```

Das Hinzufügen von Datensätzen zum vorherigen Beispiel füllt die Tabelle mit Daten:

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro
Zeit,Flux,Temperatur
[Sekunden seit 1.1.1970],[Jy],[°C],@
1073217600.370,2602,-2.4,1073217600.590,1
1073217600.390,2595,-2.4,1073217600.615,2
1073217600.410,2594,-2.3,1073217600.640,3
```

Insgesamt führt dies zur folgenden Datenstruktur:

```
0             EKD@JO63rx_Dambeck.RSpectro
0-0             Zeit
0-0-0             [Sekunden seit 1.1.1970]
0-0-0-0             1073217600.370
0-0-0-1             1073217600.390
0-0-0-2             1073217600.410
0-1             Flux
0-1-0             [Jy]
0-1-0-0             2602
0-1-0-1             2595
0-1-0-2             2594
0-2             Temperatur
0-2-0             [°C]
0-2-0-0             -2.4
0-2-0-1             -2.4
0-2-0-2             -2.3
0-3             <leer>
0-3-0             @
0-3-0-0             1073217600.590
0-3-0-1             1073217600.615
0-3-0-2             1073217600.640
0-4             <leer>
0-4-0             <leer>
0-4-0-0             1
0-4-0-1             2
0-4-0-2             3
```

### Volatiles Synchronschreiben

Wenn Datensätze lediglich einen Augenblickswert besitzen und zum Beispiel nur zur dynamischen Anzeige dienen, dann können sie als „volatil“ eingestuft werden. In diesem Fall wird keine Speicherung vorgenommen. Ein Beispiel wäre die Übertragung der Uhrzeit, bei der es in der Regel wenig sinnvoll wäre, jeden neuen Sekundenwert zu speichern. Andererseits kann bei Videoübertragungen das Speichern des Datenstromes wegen des großen Speichervolumens, aus Gründen einer geringen Relevanz oder aus rechtlichen Gründen nicht durchführbar sein. Dennoch wird es jedoch in der Regel technisch sinnvoll sein, eine begrenzte Anzahl zurückliegender Datensätze vorzuhalten, um diese z.B. bei Übertragungsstörungen neu übertragen zu können.

Die genannten Anforderungen werden durch einen Ringpuffer mit Speicherplätzen für N Datensätze erfüllt. Die Anzahl der Datensätze im Ringpuffer kann 0 oder größer sein und wird bei synchronen Schreiboperationen im Feld nach dem '@'-Zeichen angegeben. Bei 0 erfolgt kein Vorhalten von

Datensätzen sondern ein direktes Überschreiben der vorherigen Daten und der Ringpuffer ist in diesem Fall nicht existent. Falls jedoch, so wie zuvor beschrieben, keine Größe des Ringpuffers spezifiziert wurde dann werden alle Datensätze gespeichert.

Wenn alle Datensätze gespeichert werden (nur '@'), dann ist in der Regel kein Anfügen einer Reihenfolgenummer bei den Datensätzen erforderlich, weil diese sich implizit aus der Datenstruktur ergibt. Falls in diesem Fall unterhalb der '@'-Position nur ein Wert erscheint, dann wird dieser daher als Systemzeitstempel des Speicherns im gleichen Format wie der Zeitstempel am Zeilenanfang interpretiert. Bei Angabe der Größe eines Ringpuffers und nur einem Wert unterhalb der '@'-Position wird dies jedoch als Reihenfolgenummer des Speicherns im Ringpuffer interpretiert.

Beispiel für einen Übertragungszeitpunkt mit Ringpuffer und Reihenfolgenummer:

```
EKD@JO63rx_Dambeck.RSpectro
Zeit,Flux,Temperatur
[Sekunden seit 1.1.1970],[Jy],[°C],@,3
1073217600.410,2602,-2.4,129
1073217600.370,2595,-2.4,127
1073217600.390,2594,-2.3,128
```

Der Ringpuffer umfasst 3 Datensätze. Im ersten Datensatz befinden sich die zuletzt erfassten Daten. Im Datensatz 2 liegen die ältesten Daten vor, welche als nächstes überschrieben werden.

### Wiederaufsetzen beim Synchronschreiben

Wenn ein Synchronschreiben nach dem Schreiben anderer Daten an einem vorherigen Punkt wieder neu aufgesetzt werden soll, dann wird dazu im ersten Feld einer Zeile die Adresse des '@'-Zeichens vom unterbrochenen Synchronschreiben und optional eine Ringpufferlänge angegeben. Zusätzlich kann in einem weiteren Feld eine von 0 abweichende Position zum Fortsetzen des Synchronschreibens angegeben werden, z.B.

```
0-3-0,3,130
```

setzt das Schreiben der obenstehenden Tabelle an Position 130 im dreizeiligen Ringpuffer fort.

### Datentypen am Zeilenanfang

Beim Schreiben auf einem aktuellen Pfad und allgemein beim synchronen Schreiben von Datensätzen fehlt am Zeilenanfang eine Kennzeichnung für den Datentyp. Dieser leitet sich daher implizit aus folgenden Regeln ab:

- beim synchronen Schreiben wird der Datentyp des übergeordneten Elementes übernommen
- wenn kein synchrones Schreiben aktiv ist, dann stellt ein Adressformat eine Adresse dar
- wenn kein synchrones Schreiben aktiv ist, dann stellt ein numerisches Format eine Zahl dar
- wenn keiner der aufgeführten Fälle zutrifft dann stellt der Feldinhalt einen Text dar

Aus den genannten Regeln leitet sich insbesondere die Einschränkung ab, dass ein binärer Inhalt am Zeilenanfang nur beim synchronen Schreiben erkannt wird. Wenn dagegen ein binärer Inhalt am Zeilenanfang beim Schreiben einer Informationsmenge auf einem Pfad auftritt, dann muss eine explizite Kennzeichnung des Datentyps erfolgen, z.B. durch Voranstellen einer Adresse für den aktuellen Pfad, gefolgt von einem Gleichheitszeichen als Kennung für das Binärformat.

### Tabellen-Import

Die strukturierten Daten aus den vorherigen Beispielen können in ein Tabellenprogramm importiert werden und führen zur nachfolgend gezeigten Ansicht. Somit wurde das Entwurfsziel erreicht, dass die Anzeige von strukturierten Daten eines Files/Streams in lesbarer Form erfolgen kann. Weiterhin sind alle Daten somit auch einer direkten Weiterverarbeitung in Tabellenform zugänglich:

| EKD@JO63rx_Dambeck.RSpectro | | |
|---|---|---|
| Zeit | Flux | Temperatur |
| [Sekunden seit 1.1.1970] | [Jy] | [°C] |
| 1073217600.370 | 2602 | -2.4 |
| 1073217600.390 | 2595 | -2.4 |
| 1073217600.410 | 2594 | -2.3 |

@

1073217600.590 1

1073217600.615 2

1073217600.640 3

## Vergleich von FTLight-Strukturen mit Verzeichnissen, Registraturen und Datenbanken

Nach der Einführung von Informationsmengen kann die Frage auftauchen, welchen Unterschied es zwischen FTLight-Informationsmengen und den damit zusammenhängenden hierarchischen Strukturen einerseits und den gut bekannten Informationsstrukturen wie Verzeichnissen in Dateisystemen, Registraturen und Datenbanken andererseits gibt. Eine kurze Antwort wäre, dass mit der FTLight-Spezifikation künstliche Beschränkungen beseitigt werden, welche den anderen genannten Datenstrukturen vorwiegend aus Performance-Gründen und zum Zweck eines effektiven Ressourcen-Management auferlegt wurden.

Die FTLight-Spezifikation ist ebenfalls auf Performance und effektives Ressourcen-Management ausgerichtet. Jedoch wird mit höchster Priorität eine konzeptionelle Grenzenlosigkeit angestrebt. Beschränkungen werden erst dann sichtbar, wenn eine FTLight-Datei in ein Dateisystem geschrieben wird oder wenn ein FTLight-Stream zwischen Computern übertragen wird. Die in solchen praktischen Anwendungen auftretenden Beschränkungen resultieren aus der endlichen Größe des zur Verfügung stehenden Speicherplatzes beziehungsweise aus der begrenzten Bandbreite des Übertragungskanals zwischen Computern, der den FTLight-Stream überträgt.

Im folgenden werden einige Fallbeispiele betrachtet, wie die FTLight-Spezifikation Schranken überwindet:

### Verzeichnisse in Dateisystemen

Gewöhnlich werden von jedem Dateisystem Beschränkungen bezüglich der Größe von Pfadkomponenten sowie bezüglich der Zeichen gemacht, welche in Pfadkomponenten vorkommen dürfen. Zusätzlich unterliegt oftmals die Gesamtlänge eines Pfades einer Beschränkung durch das Dateisystem oder durch das Betriebssystem, welches das Dateisystem anwendet. Eine übliche Beschränkung für die Gesamtlänge eines Pfades ist zum Beispiel 1024 Byte, wobei solche Zeichen wie Slash ‚/' oder Backslash ‚\' nicht innerhalb von Pfadkomponenten vorkommen dürfen.

Die genannten Beschränkungen werden in der FTLight-Spezifikation überwunden. Die Komponenten eines Pfades wie auch die Gesamtlänge eines Pfades sind konzeptionell unbeschränkt. Weiterhin sind gleich zwei Datentypen verfügbar (Text und Binär), welche das Erzeugen von Pfadkomponenten aus beliebigen 1-Byte-Zeichen 0..255 sowie auch aus beliebigen Bit-Feldern als Bestandteil eines Pfades ermöglichen.

### Registraturen

Grundsätzlich gibt es keine großen konzeptionellen Unterschiede zwischen einem Eintrag in einer Registratur (z.B. Windows Registry) und einer Datei in einem Dateisystem außer zusätzlicher Beschränkungen bezüglich der Größe von Einträgen in einer Registratur. Eine gemeinsame Beschränkung beider Strukturen besteht darin, dass auf einem gegebenen Pfad jeder Datei beziehungsweise jedem Namen einer Registratur immer nur ein Inhalt zugewiesen werden kann. Sobald mehrere Werte zugewiesen werden sollen muss die Struktur immer erst um eine neue Ebene ergänzt werden.

In einer Registratur würde die zusätzliche Ebene aus mehreren Einträgen bestehen, wobei jeder Eintrag einen eigenen eindeutigen Namen aufweisen muss. Anschließend können die Werte den

Namen zugeordnet werden. Eine äquivalente Vorgehensweise in einem Dateisystem wäre das Anlegen von Dateien mit unterschiedlichen Dateierweiterungen (Extensions), zum Beispiel .exe für ausführbare Dateien und .ini für die Initialisierungsinformation während die Dateinamen vor der Extension gleich wären. Dies wäre jedoch nur eine Umgehungslösung für die genannte Beschränkung.

Die FTLight-Spezifikation überwindet die Einschränkung, dass einem Namen in einer Registratur oder einem Dateinamen nur ein einziger Wert zugewiesen werden kann durch den Ansatz, dass jedem Pfad eine konzeptionell unbegrenzte Menge von Informationselementen zugewiesen werden kann. Das ermöglicht zum Beispiel das Verwalten aller Messwertdaten unter dem Namen der Datenquelle, welche diese Werte generiert hat. Der Vorteil gegenüber dem Abspeichern aller Messwertdaten in einer Datei besteht darin, dass jeder einzelne Messwert ein eigenständiges Informationselement unter dem Dach einer gemeinsamen Spezifikation darstellt und dass somit keine zusätzlichen Regeln für das Schreiben/Lesen solcher Werte in/aus Dateien erforderlich sind.

**Man kann die FTLight-Spezifikation somit als Vereinheitlichung der traditionellen Konzepte von Pfad, Verzeichnis und Dateien mit den unterschiedlichsten Dateiformaten in einem gemeinsamen Konzept ansehen, wo Informationsmengen mit anderen Informationsmengen verknüpft werden mit der Absicht, unbeschränkte hierarchische Informationsstrukturen zu schaffen.**

**Datenbanken**

Relationale Datenbanken sind in der Regel hoch-effektiv beim Verwalten von Tabellen-artigen Daten, welche durch einen Satz von Regeln zwischen den Tabellen beschrieben werden können. Dagegen sind sie für das Verwalten von hierarchisch organisierten Daten oftmals weniger gut geeignet.

Die FTLight-Spezifikation erweitert das Konzept hierarchischer Datenstrukturen durch die Fähigkeit, Tabellen-artige Daten in synchronisierten geordneten Informationsmengen zu verwalten, zum Beispiel im Falle von Messwertserien mit einem Zeitstempel als Schlüssel von Messwertsätzen.

# Datentypen

Die folgenden Datentypen erlauben es, die Informationselemente in der jeweils zutreffendsten Art darzustellen. Die Definition erfolgte so, dass ein Parser bei der Verarbeitung eines FTLight-Streams die einzelnen Elemente eindeutig identifizieren kann:

- Text (Single-Byte-Zeichen von 0..255, unbegrenzte Anzahl)
- Zahlen (Alle Zahlenformate, die ausschließlich folgende Zeichen verwenden: “0123456789 - + . E e X x A a B b C c D d F f”)
- Binär (Alle Bit-Felder, angefangen von Einzelbit bis zu unbegrenzter Bit-Anzahl)
- Identifikator (enthält Informationen zum Operator, dem Ort sowie zu einem Thema)
- Adresse (ermöglicht das Referenzieren von Elementen innerhalb eines Files/Streams sowie das Herstellen von Verbindungen [Links])

## Sonderzeichen

Die folgenden Zeichen haben eine spezielle Bedeutung innerhalb von FTLight Files/Streams und müssen daher in allen Datentypen, wo Verwechselungen möglich sind vermieden werden bzw. ihre Sonderbedeutung muss mittels vorangestelltem Backslash entwertet werden.

- 10 (Zeilenschaltung) - Zeilentrenner
- 13 (Wagenrücklauf) - Zeilentrenner
- 44 (Komma) - Feldseparator
- 45 (Minuszeichen) - Adresselement
- 58 (Doppelpunkt) - Feldseparator
- 59 (Semikolon) - Feldseparator
- 61 (Gleichheitszeichen) - Feldseparator
- 64 (@ Zeichen) - Identifikator, Operator
- 96 (Back-Apostroph) - Anfrageoperator/„Leere“ (QUERY, EMPTY)
- 127 (Löschen) - Löschoperator (DEL)

## Textdatentyp

Der Textdatentyp kann alle Einzelbyte-Zeichen von 0..255 darstellen. Für die Vermeidung von Konflikten mit den Sonderzeichen müssen diese wie zuvor beschrieben in ihrer Sonderbedeutung entwertet werden, was durch Voranstellen eines Backslash ‘\’ bewirkt wird. Beim Verarbeiten eines Textes muss das Backslash-Zeichen vor Sonderzeichen wieder entfernt werden.

Die Möglichkeit der Darstellung sämtlicher Einzelbyte-Zeichen ermöglicht es grundsätzlich, mit dem Textdatentyp beliebige Daten einschließlich Binärdaten darzustellen. Jedoch sollte man sich bewusst sein, dass Dateien mit dieser Darstellung möglicherweise nicht mit normalen Texteditoren geöffnet werden können, weil sie unter anderem auch die Steuerzeichen (0..31) enthalten können.

Weiterhin kann bei angenommener Gleichverteilung aller Zeichen im Text durch das Hinzufügen eines Backslash zu den Sonderzeichen insgesamt mit einer Effektivität der Kodierung von etwa 96% gerechnet werden. Aus diesem Grunde sollte dem Binärdatentyp der Vorrang gegeben werden, welcher eine Effektivität der Kodierung von 97% erreicht. Weiterhin werden mit dem Binärdatentyp alle Sonderzeichen (0..31) bereits vom Design her vermieden..

Beispiele:

Dies ist ein Beispiel für den Textdatentyp\: “mail\@server.com”.

## Datentyp für Zahlendarstellung

Sobald ein als Text/Zahl deklariertes Feld ausschließlich die Zeichen “0123456789 - + . E e X x A a B b C c D d F f” enthält wird es weiter analysiert, ob es einer gültigen Zahlendarstellung entspricht. Falls dies der Fall ist, dann wird das Feld in eine Zahl konvertiert. Die Länge einer Zahl unterliegt keinerlei Beschränkungen.

Die folgenden Zahlendarstellungen werden akzeptiert:

- **Integer** – jede Kombination der Ziffern 0..9, z.B. 0, 10, 938776658832671414423574758
- **Fließkomma** – zwei Integer-Zahlen, verbunden mit einem Punkt, z.B. 123.4, 0.56, .87, 543.
- **Wissenschaftlich** – Fließkomma zuzüglich Exponent, z.B. 0.56E-2, 0.62e12, 4.283E+5
- **Hexadezimal** – ‘0x’ oder ‘0X’ gefolgt von Hexadezimalzeichen 0..9,A..F,a..f, z.B. 0x03FC

## Arrays von Zahlen und Text

Für das Kodieren von Zahlen/Text-Arrays wird das Back-Apostroph (ASCII-Code 96) verwendet. Wenn dieses in einem Zahlen/Text-Feld ohne Entwertung durch Backslash-Zeichen erscheint und nicht direkt von einem Feldseparator gefolgt wird, dann leitet es ein Array von Zahlen/Text ein. Die Anzahl der Back-Apostrophs steuert dabei die Ebene in den Array-Dimensionen, z.B:

```
EKD@JN58nc.Array,0
,Zahlen`1,2,3
,Text`A,B,C
```

Das Array '`Zahlen`1,2,3`' führt auf eine ähnliche hierarchische Struktur wie bei Verwendung eines Doppelpunktes '`Zahlen:1,2,3`' zum Start einer neuen Informationsmenge, jedoch auf der nächst tieferen untergeordneten Ebene. Weiterhin können mit dem Back-Apostroph im Unterschied zum Doppelpunkt beliebig viele Ebenen von Array-Dimensionen durch die Anzahl der aufeinander folgenden Back-Apostrophs adressiert werden, z.B. ein zweidimensionales Array:

```
EKD@JN58nc.Array,0
,Zahlen``1,2,3,`4,5,6,`7,8,9
```

Obenstehend wird durch ein doppeltes Back-Apostroph ein zweidimensionales Array eingeleitet. Anschließend folgen drei eindimensionale Arrays, welche die Zeilen des zweidimensionalen Arrays darstellen. Die Bereiche der Indizes ergeben sich jeweils aus der maximalen Anzahl von Elementen.

Die Verwendung des Back-Apostrophs (=QUERY/EMPTY-Operator) als Trennzeichen am Anfang des Arrays ordnet das gesamte Array dem „Leere“-Index innerhalb der Informationsmenge zu, im Gegensatz zu den üblichen Indizes 0,1,..,N für alle anderen Elemente der Informationshierarchie.

Wenn eine Adressangabe von Back-Apostrophs eingeschlossen ist dann wird dies zur Anfrage nach dem adressierten Element des Arrays, bzw. zur Anfrage nach dem adressierten Teil (Untermenge) des vollständigen Arrays, z.B.:

```
EKD@JN58nc.Array,Zahlen`2-1`
```

gibt nur das Element „8“ von der Position 2-1 aus dem oben definierten Array zurück, während

```
EKD@JN58nc.Array,Zahlen`1`
```

die Zeile 4,5,6 zurück gibt.

Arrays unterschiedlicher Dimensionen an der gleichen Position beeinflussen sich gegenseitig nicht und können daher unabhängig voneinander geschrieben und gelesen werden.

## Binärer Datentyp (BinX)

Das Kodieren von Binärdaten basiert auf einem Satz von Symbolen, welche die Werte von 0..215 repräsentieren. Diese Symbole (0..215) werden in FTLight Files/Streams in solcher Weise den verfügbaren Zeichen (0..255) zugeordnet, dass kein Steuerzeichen (0..31) und keines der festgelegten Sonderzeichen für die Darstellung eines Symbols benötigt wird.

Die Wandlung von Symbolen in erweiterte ASCII-Zeichen geschieht in folgender Weise:

| | | |
|---|---|---|
| Symbol = 12 | → Zeichen = 248 | (vermeiden von 44 - Komma) |
| Symbol = 13 | → Zeichen = 249 | (vermeiden von 45 - Minuszeichen) |
| Symbol = 26 | → Zeichen = 250 | (vermeiden von 58 - Doppelpunkt) |
| Symbol = 27 | → Zeichen = 251 | (vermeiden von 59 - Semikolon) |
| Symbol = 29 | → Zeichen = 252 | (vermeiden von 61 - Gleichheitszeichen) |
| Symbol = 32 | → Zeichen = 253 | (vermeiden von 64 - @ Zeichen) |
| Symbol = 64 | → Zeichen = 254 | (vermeiden von 96 - Back Apostroph) |
| Symbol = 95 | → Zeichen = 255 | (vermeiden von 127 - Löschen) |
| alle anderen | → **Zeichen = Symbol + 32** | |

In der Gegenrichtung von einem Zeichen zum Symbol geschieht die Konvertierung wie folgt:

| | |
|---|---|
| Zeichen 32..247 | → **Symbol = Zeichen – 32** |
| Zeichen 248 | → Symbol = 12 |
| Zeichen 249 | → Symbol = 13 |
| Zeichen 250 | → Symbol = 26 |
| Zeichen 251 | → Symbol = 27 |
| Zeichen 252 | → Symbol = 29 |
| Zeichen 253 | → Symbol = 32 |
| Zeichen 254 | → Symbol = 64 |
| Zeichen 255 | → Symbol = 95 |

Vier aufeinanderfolgende Symbole aus einem Binärdatenfeld werden mit einem Radix von 216 zu einem 31-Bit-Feld kombiniert. Falls mehr als 31 Bits im Binärdatenfeld enthalten sind dann wird ein Vielfaches von 4 Zeichen für die Kodierung verwendet, bzw. es werden ein, zwei oder drei Zeichen angefügt, solange bis die Länge des Bit-Feldes erreicht ist. In dieser Weise können Binärdaten beliebiger Länge wie z.B. auch Grafiken, Sound- und Videodateien kodiert werden.

Beispiel:

Das Binärdatenfeld ‘ABCD’ wird in folgende Symbole übersetzt:

A → 65-32=**33** B → 66-32=**34** C → 67-32=**35** D → 68-32=**36**

Daraus ergibt sich für den Wert des Bit-Feldes:

(((**33** * 216 + **34**) * 216) + **35**) * 216 + **36** = **334157868**

was äquivalent zum folgenden Bit-Feld ist: **00100111110101011011000001011 00**

Das gewählte Kodierschema repräsentiert ein 31-Bit-Feld durch vier Zeichen des erweiterten ASCII-Zeichensatzes. Dies entspricht einer 31/32-Effizienz oder etwa 97% im Vergleich zum reinen Binärformat.

Der verfügbare Wertebereich für eine Gruppe von vier Symbolen entspricht

216^4 - 1 = 2,176,782,335.

Im Vergleich zum, mit diesen Symbolen dargestellten, Maximalwert eines 31-Bit-Feldes von 2,147,483,647 existieren somit insgesamt 29,298,688 ungenutzte Kombinationen.

Offene Aufgabe:

> Die Überschusskombinationen bieten eine gute Gelegenheit für erweiterte Funktionen, wie z.B. Möglichkeit für beliebige Bit-Zahlen, Kompression von zusammenhängenden Null-Bit oder Eins-Bit-Feldern sowie Kompression von sich wiederholenden komplexen Strukturen.

## Zahlendarstellung im Binärformat (NumX)

Beim Umgang mit Zahlen haben sich zweckmäßige, lesbare Formen in Abhängigkeit vom Anwendungsfall herausgebildet, z.B. Integer-Zahlen für Zählvorgänge oder die Exponentialschreibweise im wissenschaftlichen Bereich mit einem Bedarf an effektiver Darstellung auch von sehr großen oder sehr kleinen Zahlen. Diesen Anforderungen soll in FTLight durch eine Definition von gängigen Zahlenformaten entsprochen werden.

Das Repräsentieren von Zahlen erfordert die folgenden Elemente:

- Integer-Zahlen
- Negative Zahlen
- Brüche von Integer-Zahlen (rationale Zahlen)
- Exponent
- Basis zu einem Exponenten
- Negativer Exponent

Zunächst wird der Basistyp des Binärformats (BinX) für das Kodieren beliebig großer Integer-Zahlen verwendet. Das Back-Apostroph (96) wird anschließend zur Strukturierung eines Binärdatenfeldes in mehrere Komponenten entsprechend der Topologie nach folgendem Muster verwendet:

| | |
|---|---|
| BinX | - Integer-Zahl, z.B. 123 |
| `BinX | - **-BinX**, negative Zahl (hier: Integer-Zahl), z.B. -123 |
| BinX` | - **1 / BinX**, Bruch, z.B. 1/123 |
| BinX1`BinX2 | - **BinX1 * BinX2**, Produkt von Integer-Zahlen, z.B. 123*10=1230 |
| BinX1`BinX2` | - **BinX1 / BinX2**, Bruch von Integer-Zahlen, z.B. 123/10=12.3 |
| BinX1`BinX2`BinX3 | - **BinX1 * BinX2 exp BinX3**, wissenschaftlich, z.B. 123E+4 |
| BinX1`BinX2``BinX3 | - **BinX1 * BinX2 exp -BinX3**, wissenschaftlich, z.B. 123E-4 |
| BinX1`BinX2`BinX3`BinX4 | - **BinX1 / BinX2 * BinX3 exp BinX4**, wissenschaftlich, 1.3E+4 |
| BinX1`BinX2`BinX3``BinX4 | - **BinX1 / BinX2 * BinX3 exp –BinX4**, wissenschaftlich, 1.3E-4 |

## Repräsentieren von Datentypen im Binärformat (TypeX)

Anforderungen

Es ist üblich, dass jede Softwareanwendung ein spezifisches Datenformat verwendet, sobald Daten persistent auf ein Speichermedium geschrieben werden bzw. wenn Daten von einer Instanz der Anwendung zu einer zweiten Instanz der Anwendung übertragen werden. Diese Datenformate werden von Entwicklern einer Software basierend auf den gestellten Anforderungen in der Regel derart festgelegt, dass die Daten in möglichst effizienter Weise übertragen und gespeichert werden können.

Oftmals treffen diese einmal erfolgten Festlegungen zu einem späteren Zeitpunkt auf Hindernisse, wenn die auf den ursprünglichen Anforderungen basierenden Formate für das Einführen neuer Eigenschaften nicht mehr ausreichen und daher die Festlegungen erweitert oder eventuell auch

geändert werden müssen. Vom Benutzerstandpunkt aus geht die Software dabei während ihres Lebenszyklus durch verschiedene Versionen, wobei die Verträglichkeit der Datenformate von aufeinanderfolgenden Softwareversionen eine der größten Herausforderungen für die Entwickler darstellt. In der Vergangenheit gab es dabei oftmals keine Möglichkeit für eine Rückwärts-Kompatibilität, wodurch der einzige Ausweg im Festlegen eines neuen Datenformates bestand und wo Forderungen nach Verarbeitbarkeit älterer Formate durch geeignete Konverter gelöst wurden.

Konzept

Obwohl die FTLight-Spezifikation ihrerseits als offener Container für das Speichern und Übertragen beliebiger Datenstrukturen in effizienter Weise entwickelt wurde, so kann es dennoch sinnvoll sein, vorhandene Dateien durch einen FTLight-Container zu umschließen, z.B. um von der flexiblen Metadaten-Darstellung in FTLight zu profitieren. Gleichzeitig können die mit aufeinanderfolgenden Versionen in Zusammenhang stehenden Probleme transparent gelöst werden, indem die entsprechenden FTLight-Elemente wie eindeutige Identifikatoren und interne Referenzen in Verbindung mit dem Update-Mechanismus zum Einsatz kommen. Weiterhin kann auch die Einführung unterschiedlicher Binärformate empfehlenswert sein, wenn dadurch spezifische Anforderungen wie z.B. zur Verarbeitungsgeschwindigkeit oder zur Datenkompression besser gelöst werden, als wie dies mit der standardmäßigen FTLight-Repräsenation von Binärdaten der Fall ist . Die folgenden Datenelemente gestatten es daher, andere Datenformate (extern zu FTLight) zu kapseln wie auch die eingebauten Formateigenschaften in transparenter Weise zu erweitern.

**Identifikatoren für Datentypen**

Die ersten vier BinX-Symbole in einem binären Datenfeld dienen als Typidentifikator. Sobald ein Datentyp in einem binären Datenfeld erkannte wurde wird dieser auch auf alle untergeordneten Binärdatenfelder übertragen. Alle Type-X-Datenfelder werden als little endian übertragen.

Datentypen können kaskadiert werden, wie zum Beispiel BinXvalue und BinXbinary. Im genannten Fall wird mit BinXvalue eine Skalierung aller nachfolgenden Binärwerte bezogen auf eine Einheit festgelegt und BinXbinary kann anschließend ein mehrdimensionales Array der Werte definieren.

Ein Typidentifikator benutzt solche Kombinationen von FTLight-Symbolen, welche nicht bereits zur Darstellung von 31-Bit-Feldern durch vier Zahlen mit einem Radix von 216 Verwendung finden. Falls stattdessen die ersten vier Symbole bereits ein gültiges 31-Bit-Datenfeld darstellen, dann handelt es sich um ein herkömmliches binäres BinX-Datenfeld ohne Typinformation.

Die folgenden TypeX-Identifikatoren sind basierend auf dem Maximalwert von

BinXmax = 216^4 – 1 = 2,176,782,335 absteigend definiert:

| | | | |
|---|---|---|---|
| **FTLightOpen** | = BinXmax - 0 | = 2,176,782,335 | - Offenes Format mit FTLight-Symbolen |
| **FTLightWrap** | = BinXmax - 1 | = 2,176,782,334 | - Beliebiges Format mit BinX gekapselt |
| **BinMCL** | = BinXmax - 2 | = 2,176,782,333 | - BinMCL-kodiertes Bitfeld-Array |
| **BinXbinary** | = BinXmax - 3 | = 2,176,782,332 | - BinX-kodiertes Bitfeld-Array |
| **BinXstring** | = BinXmax - 4 | = 2,176,782,331 | - BinX-kodiertes String-Array |
| **BinXvalue** | = BinXmax - 5 | = 2,176,782,330 | - BinX-kodierter physikalischer Wert |
| **BinXtime** | = BinXmax - 6 | = 2,176,782,329 | - BinX-kodierte Zeit |
| **CmXtoken** | = BinXmax - 7 | = 2,176,782,328 | - Token für Cosmos-Kommunikation |
| **CmXlink** | = BinXmax - 8 | = 2,176,782,327 | - Link auf lokal erreichbares Modul |
| **BinDIF** | = BinXmax - 9 | = 2,176,782,326 | - BinDIF-kodiertes Integer-Array |

## Typsteuerfelder (ControlX)

Ein TypeX-Identifikator kann von Typsteuerfeldern gefolgt sein. Die Bedeutung der Typsteuerfelder ist spezifisch und wird für jeden Typ separat beschrieben. Ein Steuerfeld ist eine unbegrenzte Zeichenfolge, welche nach BinX-Regeln kodiert ist. Die Steuerfelder sind durch Back-Apostroph (96) voneinander getrennt:

**TypeX**`BinX(n)`ControlX1`…`ControlXn-1`TypeX(Binärdaten)

Das erste Steuerfeld enthält jeweils die Gesamtanzahl aller Steuerfelder. Daher wurde es mit BinX(n) = ControlX0 benannt. Das nach den Steuerfeldern folgende TypeX(Binärdaten)-Datenfeld enthält die nach den Regeln des jeweiligen TypeX-Datentyps kodierten Daten.

Falls das Zeichen nach dem TypeX-Identifikator kein Back-Apostroph ist dann stellt der darauf folgende Binärdatenblock ein homogenes, eindimensionales Bit- oder Integer-Feld vom TypeX-Datentyp dar, ohne weitere Strukturinformationen, zumindest keine, die dem FTLight-Container bekannt wären.

## FTLightOpen-Datentyp

Der FTLightOpen-Datentyp stellt eine Einladung zum Entwickeln weiterer hochspezifischer Datenformate dar, welche auf der Basis des Radix-216-Schemas und unter ausschließlicher Verwendung von zulässigen FTLight-Zeichen arbeiten. Es wird empfohlen, dass eine Beschreibung des jeweiligen Datenformats im Internet erfolgt und dass eine URL zu dieser Beschreibung in ControlX1 = URL gegeben wird:

**FTLightOpen**`BinX(n)`BinX(URL)`ControlX2`…`ControlXn-1`FTLight(Binärdaten)

Die Anzahl der Steuerfelder, welche auf die URL folgen, ist spezifisch für das jeweilige Datenformat. Falls nach dem FTLightOpen-Identifikator kein Back-Apostroph folgt dann gilt für das sich anschließende Binärdatenfeld eine Kodierung im BinX-Format.

## FTLightWrap-Datentyp

Der FTLightWrap-Datentyp dient zum Kapseln von Datenformaten, die bereits außerhalb von FTLight bestehen sowie zum Kapseln kompletter FTLight-Archive. Es wird empfohlen in ControlX1 = URL einen Link zu einer Website von einem Programm zu geben, welches das gekapselte Datenformat verarbeiten kann oder alternativ den Namen eines Programms, falls ein Link nicht verfügbar ist.

**FTLightWrap**`BinX(n)`BinX(URL/Programm)`ControlX2`…`ControlXn-1`BinX(Binärdaten)

Die Anzahl der Steuerfelder ist spezifisch für das jeweilige Format. Falls nach dem FTLightWrap-Datentyp keine Steuerfelder folgen (kein Back-Apostroph nach dem Typidentifikator) dann wird von einem kompletten FTLight-Archiv ausgegangen..

## BinMCL-Datentyp

Der dem BinMCL-Datentyp von den Eigenschaften her ähnliche BinX-Datentyp erfordert für das Berechnen der aufeinanderfolgenden BinX-Zeichen mehrere arithmetische Operationen. Abhängig vom Prozessor kann dies zu einem beträchtlichen Zeitbedarf für das Erzeugen des BinX-Datenstromes führen. Daher ist es für extreme Laufzeitanforderungen erforderlich eine solche Kodierung zu verwenden, welche ausschließlich durch das Verschieben von Bits und unter Verwendung von Lookup-Tabellen auskommt, ohne dabei aufwendige arithmetische Operationen wie Multiplikation oder Division durchführen zu müssen. Dieses Format kann zum Beispiel auf Computern mit einer geringen Taktrate zur Anwendung kommen, wo dennoch ein hoher Durchsatz erzielt werden soll. Es eignet sich jedoch ebenso für schnelle Rechner, wenn extreme Anforderungen bezüglich der zu erreichenden Datenrate erfüllt werden sollen.

Eine solche Kodierung, welche die beschriebenen Anforderungen erfüllt, wurde von Marko Cebokli als ein 15/16-Format entwickelt. Es arbeitet auf der Basis von 30-Byte-Feldern oder besser 15 Feldern mit jeweils 16 Bit. Zunächst wird das höchstwertige Bit (MSB) eines jeden Datenwortes schrittweise in ein 16-Bit-Register eingeschoben. Anschließend erfolgt das Nachschlagen in einer vorausberechneten Tabelle basierend auf dem jeweiligen 16-Bit-Wert ohne das höchstwertige Bit als 15-Bit-Indexwert. Diese Tabelle wurde zuvor mit den zugeordneten FTLight-Zeichen für die insgesamt 32768 15-Bit-Werte gefüllt. Im letzten Schritt wird das Register, welches alle MSB derart aufgenommen hat, dass das MSB des letzten 16-Bit-Wertes an der niederwertigsten Position (LSB) ist, ebenfalls als 15-Bit-Wert in der vorausberechneten Tabelle nachgeschlagen.

Die Implementierung eines 15/16-Algorithmus ergibt eine Effizienz der Kodierung von 93.8% im Vergleich zu den 96.9% eines 31/32-Algorithmus wie BinX. Der Vorteil eines 15/16-Algorithmus bei Implementierung mit einer Lookup-Tabelle besteht jedoch zum Beispiel in einem Bedarf von nur 42 Prozessortakten pro Byte auf einem Pentium 4 Prozessor während ein 31/32-Algorithmus beim gleichen Prozessor 135 Takte pro Byte benötigt, wodurch der 15/16-Algorithmus auf dem genannten Prozessor dreimal so schnell als wie der 31/32-Algorithmus abläuft. Diese Relationen werden sich jedoch mit fortschreitender Prozessortechnik ändern und müssen dann neu bewertet werden, um das effektivste Verfahren für einen Anwendungsfall auszuwählen.

Das Layout von BinMCL ist ähnlich dem vom BinXbinary-Datentyp:

**BinMCL**`BinX(n)`BinX(Dimension_1)`…` BinX(Dimension_n-1)`BinMCL(Binärdaten)

Als Beispiel wird ein Array mit zwei Kanälen eines Ein-Bit-Interferometers, welches Daten von zwei gleichartigen Geräten enthält folgendermaßen aussehen:

**BinMCL**`BinX(4)`BinX(1)`BinX(2)`BinX(2)`BinMCL(Binärdaten)

Ein Messpunkt besteht aus insgesamt vier Bit, wobei zwei Bit das I- und Q-Signal des ersten Gerätes und zwei weitere Bit das I- und Q-Signal des zweiten Gerätes darstellen.

Es wird offensichtlich, dass in diesem Fall sogar jedes einzelne Bit seine eigene Adresse bekommen hat und unter dieser dann auch eindeutig angesprochen werden kann. Dies ist in den Fällen von Bedeutung, wenn Datenströme von verteilten Beobachtungsstationen zu einem Korrelator-Standort übertragen und dort kombiniert (korreliert) werden sollen.

## BinXbinary-Datentyp

Der BinXbinary-Datentyp entspricht dem BinX-Format soweit keine Steuerfelder vorhanden sind. Bei Vorhandensein von mehr als einem Steuerfeld stellen diese die Dimensionen eines Arrays von Bit-Feldern dar. Das erste Steuerfeld gibt die Anzahl der Dimensionen des Arrays an:

**BinXbinary**`BinX(n)`BinX(Dimension_1)`…` BinX(Dimension_n-1)`BinX(Binärdaten)

Als Beispiel wird ein Array mit 12-Bit-Messwerten von einem 100-Kanal-Spektrometer folgendermaßen aussehen:

**BinXbinary**`BinX(3)`BinX(12)` BinX(100)`BinX(Binärdaten)

Das vorliegende Binärfeld wird als dreidimensionales Bit-Feld interpretiert, wobei die ersten beiden Dimensionen auf 12 Bit für jeden einzelnen Messwert sowie auf 100 Werte pro Messwertzeile gesetzt werden. Die Größe der dritten Dimension, welche die Anzahl der Messwertzeilen bzw. die Anzahl von Frequenzdurchläufen darstellt, folgt aus der Größe des Binärfeldes.

## BinXstring-Datentyp

Der BinXstring-Datentyp gestattet das Speichern von n-dimensionalen Arrays von null-terminierten W-Bit Zeichenketten, z.B. 16-Bit Unicode-Strings.

**BinXstring**`BinX(n)`BinX(W)`BinX(Dimension_1)`…` BinX(Dimension_n-1)`BinX(Sequenz von

Zeichenketten)

Zum Beispiel wird ein Array von Übersetzungen von einer ersten Sprache in eine zweite und eine dritte Sprache mit 8-Bit Zeichenrepräsentation folgendermaßen gespeichert werden:

**BinXstring**`BinX(3)`BinX(8)`BinX(3)`BinX(Term11 Term12 Term13 … TermN1 TermN2 TermN3)

Jedem Term wird ein Null-Zeichen (ASCII-Wert 0x00, Unicode-Wert 0x0000) hinzugefügt, welches das Ende einer null-terminierten Zeichenkette anzeigt. Jeder Term in einem Tripel von Zeichenketten hat dabei die gleiche Bedeutung, jedoch in unterschiedlichen Sprachen.

## BinXvalue-Datentyp

Der BinXvalue-Datentyp ermöglicht die Darstellung eines Bruchteils oder eines Vielfachen einer Einheitsgröße sowie auch beliebige Verhältnisse von beiden, z.B. als Basiswert für Messwertdaten. Negative Werte und Exponenten werden durch doppeltes Back-Apostroph (``) gekennzeichnet.

BinXvalue`BinX(Wert)

BinXvalue`BinX(1)`BinX(Wert)

BinXvalue`BinX(2)``BinX(Negativwert)

BinXvalue`BinX(2)`BinX(Einheit)`BinX(Wert)

BinXvalue`BinX(3)`BinX(Einheit)``BinX(Negativwert)

BinXvalue`BinX(3)`BinX(Einheit)`BinX(Faktor)`BinX(Wert)

BinXvalue`BinX(4)`BinX(Einheit)`BinX(Faktor)``BinX(Negativwert)

BinXvalue`BinX(4)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)`BinX(Wert)

BinXvalue`BinX(5)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)``BinX(Negativwert)

BinXvalue`BinX(5)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)`BinX(Exponent)`BinX(Wert)

BinXvalue`BinX(6)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)`BinX(Exponent)``BinX(Negativwert)

BinXvalue`BinX(6)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)``BinX(Negativexponent)`BinX(Wert)

BinXvalue`BinX(7)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)``BinX(Negativexponent)``BinX(Negativwert)

BinXvalue`BinX(6)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)`BinX(Exponent)`BinX(Wert)

BinXvalue`BinX(7)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)`BinX(Exponent)``BinX(negat. Wert)

BinXvalue`BinX(7)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)``BinX(Negativexp.)`BinX(Wert)

BinXvalue`BinX(8)`BinX(Einheit)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)``BinX(Negativexp.)``BinX(n. Wert)

Ein relativer Wert, ausgedrückt als Integer-Zahl, kann neben einer physikalischen Einheit zusätzlich um einen Faktor, einen Teiler, eine Basis und einen Exponenten ergänzt werden, um den endgültigen physikalischen Wert darzustellen. Zum Beispiel wird ein relativer Wert von 1, der einen Wert von 1E-26 darstellt, wie folgt kodiert:

**BinXvalue**`BinX(6)`BinX(0)`BinX(1)`BinX(1)``BinX(26)`BinX(1)

Die im Beispiel verwendete Einheit von '0', kodiert als BinX(0), bedeutet eine relative Zahl ohne physikalische Einheit. Eine physikalische Messgröße muss stattdessen eine Einheit ungleich '0' erhaltenen, welche nach folgender Zuordnung als SI-Einheit oder als eine davon abgeleitete Einheit kodiert wird:

| | | | |
|---|---|---|---|
| 0 | (ohne Einheit) | - | [-] |
| 1 | Zeit | Sekunde | [s] |
| 2 | Länge | Meter | [m] |
| 3 | Masse | Kilogramm | [kg] |
| 4 | Stromstärke | Ampere | [A] |
| 5 | thermodynamische Temperatur | Kelvin | [K] |

| | | | |
|---|---|---|---|
| 6 | Stoffmenge | Mol | [mol] |
| 7 | Lichtstärke | Candela | [cd] |
| 8 | ebener Winkel | Radiant | [rad] |
| 9 | Raumwinkel | Steradiant | [sr] |
| 10 | Frequenz | Hertz | [Hz] |
| 11 | Kraft | Newton | [N] |
| 12 | Druck | Pascal | [Pa] |
| 13 | Energie, Arbeit, Wärmemenge | Joule | [J] |
| 14 | Leistung | Watt | [W] |
| 15 | elektrische Ladung | Coulomb | [C] |
| 16 | elektrische Spannung | Volt | [V] |
| 17 | elektrische Kapazität | Farad | [F] |
| 18 | elektrischer Widerstand | Ohm | [Ω] |
| 19 | elektrischer Leitwert | Siemens | [S] |
| 20 | magnetischer Fluss | Weber | [Wb] |
| 21 | magnetische Flussdichte | Tesla | [T] |
| 22 | Induktivität | Henry | [H] |
| 23 | Celsius Temperatur | Grad Celsius | [°C] |
| 24 | Lichtstrom | Lumen | [lm] |
| 25 | Beleuchtungsstärke | Lux | [lx] |
| 26 | Radioaktivität | Becquerel | [Bq] |
| 27 | Energiedosis | Gray | [Gy] |
| 28 | Äquivalentdosis | Sievert | [Sv] |
| 29 | katalytische Aktivität | Katal | [kat] |

Eine Anwendung kann von diesen Zuordnungen abweichend auch andere Binärdaten beliebiger Größe für das Kodieren einer physikalischen Einheit verwenden, welche jedoch nicht mit den bereits festgelegten Werten kollidieren dürfen.

## BinXtime-Datentyp

Der BinXtime-Datentyp erlaubt einen Bruchteil oder ein Vielfaches einer Sekunde und auch jede Kombination von beiden als Zeitbasis festzulegen. Eine negative Zeit (Vergangenheit vor 1970) wird durch ein doppeltes Back-Apostroph (96) vor der Zeitangabe ``BinX(Zeit) gekennzeichnet.

BinXtime`BinX(Zeit)

BinXtime`BinX(1)`BinX(Zeit)

BinXtime`BinX(2)``BinX(Zeit vor 1970)

BinXtime`BinX(2)`BinX(Faktor)`BinX(Zeit)

BinXtime`BinX(3)`BinX(Faktor)``BinX(Zeit vor 1970)

BinXtime`BinX(3)`BinX(Faktor)`BinX(Teiler)`BinX(Zeit)

BinXtime`BinX(4)`BinX(Faktor)`BinX(Teiler)``BinX(Zeit vor 1970)

BinXtime`BinX(4)`BinX(Faktor)`BinX(Teiler)`BinX(Exponent)`BinX(Zeit)

BinXtime`BinX(5)`BinX(Faktor)`BinX(Teiler)`BinX(Exponent)``BinX(Zeit vor 1970)

BinXtime`BinX(5)`BinX(Faktor)`BinX(Teiler)``BinX(Negativexponent)`BinX(Zeit)

BinXtime`BinX(6)`BinX(Faktor)`BinX(Teiler)``BinX(Negativexponent)``BinX(Zeit vor 1970)

BinXtime`BinX(5)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)`BinX(Exponent)`BinX(Zeit)

BinXtime`BinX(6)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)`BinX(Exponent)``BinX(Zeit vor 1970)

BinXtime`BinX(6)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)``BinX(Negativexponent)`BinX(Zeit)

BinXtime`BinX(7)`BinX(Faktor)`BinX(Teiler)`BinX(Basis)``BinX(Negativexponent)``BinX(Zeit vor 1970)

Alle Zeitangaben beziehen sich auf den 1.Januar 1970. Der Zeitschritt, welcher benötigt wird um von dieser Bezugszeit zu einem beliebigen Zeitpunkt zu gelangen wird entsprechend den zuvor für die Zahlendarstellung festgelegten Regeln angegeben. Zum Beispiel wird ein Millisekunden-Zeitschritt für positive und negative Zeiten (vor 1970) wie folgt kodiert:

**BinXtime**`BinX(3)`BinX(1)`BinX(1000)`BinX(Zeit) → Zeitangabe ab 1970-01-01 00:00:00 UTC
**BinXtime**`BinX(4)`BinX(1)`BinX(1000)``BinX(Zeit) → Zeitangabe vor 1970

## CmXtoken-Datentyp

Der CmXtoken-Datentyp überträgt ein Kommando für den Aufbau, den Abbau und die Verwaltung von „CmServiceConnection“ im Cosmos-Programmsystem. Es kommen keine Typsteuerfelder zur Anwendungen. Stattdessen wird der Definitionswert eines Kommandos als BinX-Wert direkt im Anschluss an den CmXtoken-Datentyp übertragen:

**CmXtoken**BinX(Kommando)

## CmXlink-Datentyp

Mit dem CmXlink-Datentyp werden Adressen (Funktionszeiger) von lokal erreichbaren Modulen übertragen. Es kommen keine Typsteuerfelder zur Anwendung. Stattdessen wird die Adresse direkt nach dem CmXlink-Datentyp als BinX-Datenfeld übertragen:

**CmXlink**BinX(Adresse)

## BinDIF-Datentyp

Der BinDIF-Datentyp kann ebenso wie BinMCL die Laufzeit der Datenkodierung gegenüber dem BinX-Datenformat auf einigen Rechnerarchitekturen verbessern. Weiterhin sorgt es durch die Verwendung der Differenzen von bis zu 64-Bit breiten aufeinanderfolgenden Integer-Werten neben einem geringstmöglichen Aufwand beim Kodieren und Dekodieren von Datenströmen ebenso für eine mögliche Datenkompression bei Werten mit geringer Schwankungsbreite. Deshalb eignet sich das BinDIF-Datenformat insbesondere für die Übertragung und Speicherung von Messdaten von Rechnern mit geringer Taktrate, wie bei vielen Microcontrollern anzutreffen, sowie bei fehlendem mathematischen Koprozessor.

Beim Erzeugen eines BinDIF-Wertes werden die insgesamt 216 Symbole des BinX-Datentyps durch Addition von 100 zur Differenz zum vorhergehenden Wert folgendermaßen zugeordnet:

0 – Differenz -100
1 – Differenz -99
...
99 – Differenz -1
100 – Differenz 0
101 – Differenz +1
...
199 – Differenz +99
200 – Differenz +100
201 – folgendes Symbol als absoluter Wert
202 – folgende 2 Symbole als absoluter Wert, niederwertiges Symbol zuerst
...
209 – folgende 9 Symbole als absoluter Wert, niederwertiges Symbol zuerst
210 – kein Wert (leere Position im Datenstrom)
211 – gleiche Differenz wie zuvor für die folgenden 2 Werte
...

214 – gleiche Differenz wie zuvor für die folgenden 5 Werte
215 – Beginn von jeweils 2 verschränkten Messwerten mit niedriger Schwankungsbreite

Bei den Kombinationen 201 bis 209 wird aus den sich anschließenden 1 bis 9 Symbolen zunächst ein vorzeichenloser Integer-Wert berechnet, bei aufsteigender Wertigkeit der Symbole:

**WertPositiv = (...(Symbol_9 * 216 + Symbol_8) * 216 + … ) * 216 + Symbol_1**

Anschließend wird der maximale Bereich für den Positivwert ermittelt:

**BereichPositiv = 216 ^ (Symbol_0 – 200)**

Der endgültige Wert ergibt sich aus dem Vergleich von WertPositiv mit BereichPositiv:

1) **WertPositiv < BereichPositiv / 2 : Wert = WertPositiv**

2) **WertPositiv >= BereichPositiv / 2 : Wert = WertPositiv - BereichPositiv**

Das Symbol 210 dient zum Kennzeichnen einer leeren Position in einem Datenstrom und mit den Symbolen 211 bis 214 wird eine mehrfache Wiederholung der vorherigen Differenz angezeigt.

Das Symbol 215 leitet bei Messwertfolgen mit besonders geringen Schwankungen vorrangig in den niederwertigen 4 Bits einen Bereich mit jeweils 2 verschränkten Messwerten ein, welche maximal 32-Bit breit sein können. Dieser Bereich wird beendet sobald bei einer Differenz von über 100 ein (verschränkter) Absolutwert eingefügt werden muss.

Ein verschränkter Messwert wird durch Verschieben der Bits mit einer Lücke von jeweils einem Bit erhalten. Die Bits eines zweiten Messwertes mit gleichen Bit-Lücken werden nach einer weiteren Verschiebung um ein Bit in die Lücken des ersten Wertes eingepasst und es entsteht dadurch aus zwei einzelnen Messwerten mit bis zu 32-Bit Breite ein verschränkter 64-Bit-Wert:

Wert 1: Bits in geraden Positionen: 62 – 60 - … - 14 – 12 – 10 – 8 – 6 – 4 – 2 – 0
Wert 2: Bits in ungeraden Positionen: 63 – 61 - … - 15 – 13 – 11 – 9 – 7 – 5 – 3 – 1

Ein Vorteil im Datenvolumen entsteht dadurch, dass bei Schwankungen der verschränkten Messwerte bis zu einem Wert von maximal 100 zwei Werte nur ein Byte zum Speichern benötigen.

Falls Messwertschwankungen bei maximal 16-Bit breiten Messwerten nur die niederwertigen 1 oder 2 Bit verändern dann können statt 2 Messwerten alternativ auch 4 Messwerte verschränkt werden. Dieser Modus wird durch 2 aufeinanderfolgende Symbole 215 eingeleitet und ebenfalls durch einen (verschränkten) Absolutwert beendet.

Wert 1: Bits in 0-Positionen: 60 – 56 - … - 28 – 24 – 20 – 16 – 12 – 8 – 4 – 0
Wert 2: Bits in 1-Positionen: 61 – 57 - … - 29 – 25 – 21 – 17 – 13 – 9 – 5 – 1
Wert 3: Bits in 2-Positionen: 62 – 58 - … - 30 – 26 – 22 – 18 – 14 – 10 – 6 – 2
Wert 4: Bits in 3-Positionen: 63 – 59 - … - 31 – 27 – 23 – 19 – 15 – 11 – 7 – 3

Eine Verschränkung kann auch bei 8 maximal 8-Bit breiten Messwerten angewendet werden. Dies kann zum Beispiel bei 8-Bit breiten Bilddatenströmen mit ausgedehnten Bereichen mit nur 1-Bit Schwankungen oder ohne Schwankungen eine weitere Reduktion des Datenvolumens bewirken. Dieser Modus wird durch 3 aufeinanderfolgende Symbole 215 eingeleitet und ebenso wie bei den anderen Varianten durch einen (verschränkten) Absolutwert beendet.

Wert 1: Bits in 0-Positionen: 56 – 48 – 40 – 32 – 24 – 16 – 8 – 0
Wert 2: Bits in 1-Positionen: 57 – 49 – 41 – 33 – 25 – 17 – 9 – 1
Wert 3: Bits in 2-Positionen: 58 – 50 – 42 – 34 – 26 – 18 – 10 – 2
Wert 4: Bits in 3-Positionen: 59 – 51 – 43 – 35 – 27 – 19 – 11 – 3
Wert 5: Bits in 4-Positionen: 60 – 52 – 44 – 36 – 28 – 20 – 12 – 4
Wert 6: Bits in 5-Positionen: 61 – 53 – 45 – 37 – 29 – 21 – 13 – 5
Wert 7: Bits in 6-Positionen: 62 – 54 – 46 – 38 – 30 – 22 – 14 – 6
Wert 8: Bits in 7-Positionen: 63 – 55 – 47 – 39 – 31 – 23 – 15 – 7

Beim Übertragen und Speichern von BinDIF-kodierten Daten belegt jedes Symbol ein Byte nach Wandlung des Symbols gemäß der für BinX festgelegten Zuordnung von Symbolen in Zeichen des erweiterten ASCII-Zeichensatzes. Die ersten Symbole stellen einen absoluten Anfangswert dar, gefolgt von Differenzwerten.

Sobald die Differenz zwischen zwei aufeinanderfolgenden Werten größer als 100 ist, muss statt der Differenz wieder ein Absolutwert eingefügt werden. Nach einer bestimmten Anzahl von Differenzwerten sollte zum Neustart nach Datenkorruption zur Verbesserung der Robustheit und Fehlertoleranz ebenfalls wieder ein Absolutwert eingefügt werden. Wenn dies nach zum Beispiel 31 Differenzwerten erfolgt dann ist für 8-Bit Werte die BinX Effizienz der Datencodierung erreichbar.

Das Layout von BinDIF ist ähnlich dem vom BinXbinary-Datentyp:

**BinDIF**`BinX(n)`BinX(Dimension_1)`…` BinX(Dimension_n-1)`BinDIF(Binärdaten)

Im Unterschied zum BinXbinary-Datentyp gibt es keine Festlegung für die Bitbreite eines Messwertes, weil diese bedingt durch das angewendete Differenzverfahren dynamisch zwischen 8 und 64 Bit schwanken kann. Die erste Dimension 'Dimension_1' bezieht sich daher bereits auf einen vollständigen Wert mit einer potenziell dynamischen Bitbreite.

Als Beispiel wird ein dreidimensionales Array von zwei Messgeräten mit jeweils drei Kanälen folgendermaßen aussehen:

**BinDIF**`BinX(3)`BinX(2)`BinX(3)`BinDIF(Binärdaten)

Es ist zu beachten, dass das zuvor beschriebene Differenzverfahren auf jeden Kanal jedes Messgerätes separat angewendet wird. Die dritte Dimension ergibt sich aus der Anzahl der BinDIF kodierten Daten wobei jeweils 6 Messwerte einen Datensatz bilden (2 Messgeräte mit jeweils 3 Kanälen).

## Der FTL-Identifikator-Datentyp (FTLI)

Ein FTL-Identifikator (FTLI) wird als Zeichenkette definiert, welche genau ein @-Zeichen enthält und ansonsten nur solche Zeichen, die auch ein gültiges BinX-Symbol repräsentieren können. Insbesondere darf ein Identifikator daher keine Zeichen aus dem Bereich 0..31 und auch keine der für BinX festgelegten Sonderzeichen enthalten.

Die Struktur eines Identifikators besteht somit aus einem Prefix und einem Suffix, welche durch das @-Zeichen miteinander verbunden sind. Sowohl Prefix als auch Suffix stellen in sich einen BinX-kodierten Wert dar, welcher sich bei Bedarf in jeweils einen Binärwert konvertieren und in dieser Form evaluieren lässt.

Sobald ein Identifikator als erstes Element einer Zeile in einem FTLight File/Stream erscheint so wird er zur Root (oberstes Verzeichnis) für alle nachfolgenden Informationen. Ein Identifikator stellt somit die höchste Ebene in allen Informationshierarchien dar.

Die Aufgabe eines Identifikators ist es, alle FTLight-Informationen eindeutig zu machen. Dieser Anspruch bedeutet, dass die Existenz eines doppelten Identifikators sehr unwahrscheinlich ist und dass dies für alle Daten zutrifft, welche hier auf der Erde, als auch auf anderen Planeten unseres Sonnensystems, in anderen Sonnensystemen oder in anderen Galaxien unseres Universums oder auch außerhalb unseres Universums ihren Entstehungsort haben.

Diese Spezifikation kann nur einen Vorschlag machen, wie das Problem eines eindeutigen FTLight-Identifikators gelöst werden kann. Sie ist daher als Empfehlung zu sehen, wie eindeutige Identifikatoren z.B. für das erdgebundene Sammeln von radioastronomischen Daten gebildet werden können. Der folgende Merksatz soll dabei als Wegweiser für das Generieren dienen:

> “Eine Person **A** geboren in **B** sammelt Daten in **C** zum Thema **D**”

Die vorgeschlagene Regel benutzt zum Beispiel die Anfangsbuchstaben der Komponenten A und B

in Großbuchstaben. Die Komponente C enthält eine für das globale Gradnetz geltende Locator-Bezeichnung wie sie bei Funkamateuren üblich ist und erweitert diese durch eine lokale Ortsbezeichnung. Die Komponente D enthält eine geeignete Kurzform für das Thema. Die einzelnen Elemente werden anschließend nach folgendem Schema zusammengefügt:

**AB@C.D**

Beispiel:

Der Identifikator

"EKD@**UnCmSunEar_JO63rx_Dambeck**.RSpectro"

besteht aus folgenden Komponenten:

- E - Eckhard (Vorname)
- K - Kantz (Nachname)
- D - Dambeck (Geburtsort)
- **@ - Kennzeichen für einen Identifikator**
- **Un - Standort: Universum 'Un'**
- **Cm - Standort: Kosmischer Sektor 'Cm'**
- **Sun - Standort: Sonnensystem 'Sun'**
- **Ear - Standort: Planet 'Erde'**
- **_ - Separator innerhalb der Standortbezeichnung**
- **JO63rx - Standort: Locator**
- **_ - Separator innerhalb der Standortbezeichnung**
- **Dambeck - Standort: lokale Ortsbezeichnung**
- . - Separator zwischen Ort und Thema
- RSpectro - Kurzform für "Radio Frequency Spectrometer" (Gerät als Thema)

Die Kombination von Universum, kosmischem Sektor, Sonnensystem und Planet zielt dabei auf Eindeutigkeit auf allen Ebenen. Falls diese Kombination fehlt, dann wird „UnCmSunEar" als Wert angenommen. Für die Planeten unseres Sonnensystems werden folgende Bezeichner verwendet:

UnCmSun **- Sonnensystem** (Weltraummission außerhalb von Planeten)

UnCmSun**Mer** **- Merkur**

UnCmSun**Ven** **- Venus**

UnCmSun**Ear** **- Erde**

UnCmSun**Mar** **- Mars**

UnCmSun**Pax** **- Pax** (nur als Bruchstücke im Asteroidengürtel, zu rekonstruieren)

UnCmSun**Jup** **- Jupiter**

UnCmSun**Sat** **- Saturn**

UnCmSun**Ura** **- Uranus**

UnCmSun**Nep** **- Neptun**

UnCmSun**Plu** **- Pluto**

Für den Bezeichner unseres kosmischen Sektors muss zukünftig gegebenenfalls eine weitere Strukturierung vorgenommen werden, ebenso für Orte auf Monden von einem Planeten.

Alternativ zu Zeichenketten können auch digitale Signaturen (nach entsprechender BinX-Kodierung) zu einem Bestandteil eines Identifikators werden. Sie erscheinen dann anstelle der Komponente AB und bezeichnen den Operator.

## Adressdatentyp

Die Aufgabe einer Adressangabe ist es, das Duplizieren von Informationen innerhalb eines FTLight Files/Streams durch das Referenzieren von bereits vorhandenen Informationen zu vermeiden. Der Wirkungsbereich einer Adressangabe ist dabei auf den FTLight File/Stream begrenzt, wo sie selber darin enthalten ist.

Theoretisch wäre es möglich auch Elemente in anderen FTLight Files/Streams zu referenzieren, weil die Informationen durch die verwendeten Identifikatoren auf der obersten Ebene eindeutig gemacht wurden. Trotz dieser Möglichkeit sollte kein Gebrauch davon gemacht werden, weil man sich dann auf eine fest verdrahtete Informationsstruktur verlassen würde.

Wenn man die Historie einer bereits jahrzehntelangen Repräsentation von Informationen in Datenstrukturen betrachtet so wird deutlich, dass jede Datenstruktur nach einiger Zeit veraltet und für die sich ändernden Anforderungen angepasst, erweitert oder auch vollständig neu definiert werden muss. Daher führt eine Fortsetzung von fest verdrahteten Datendefinitionen zu relativ kurzlebigen Datenstrukturen .

Im Gegensatz dazu besteht das Ziel der FTLight-Spezifikation in langlebigen Datenstrukturen, welche mit Leichtigkeit angepasst, erweitert oder sogar teilweise neu definiert werden können, ohne dabei die Verbindung zu vorher definierten Datenstrukturen abreißen zu lassen.Jedoch verlangt diese Zielstellung das Verwenden von vollständigen Pfadinformationen anstelle von internen Adressen beim Referenzieren von externen Daten.

### Adressdarstellung

Eine Adresse wird durch Integer-Zahlen dargestellt, die durch ein “-“ (Minuszeichen) miteinander verbunden sind, wobei die Folge der Integer-Zahlen die Position des entsprechenden Elementes in der Informationshierarchie innerhalb eines Files/Stream angibt. Adressen repräsentieren somit Links zu anderen Elementen, zu denen sie jeweils zeigen.

Beispiel:

0-0-1-0

Eine Adresse kann zum Beispiel als erstes Element in einer Zeile verwendet werden, falls dieses auf ein übergeordnetes Element verweisen soll, welches durch keine implizite FTLight-Regel erreicht werden kann.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,1073217600:FTLight,2004-01-12
,Antenne,Parabolspiegel 90cm
```

ist gleichbedeutend mit

```
EKD@JO63rx_Dambeck.RSpectro,1073217600
,Antenne,Parabolspiegel 90cm
0-0:FTLight,2004-01-12
```

und bezieht sich in beiden Fällen auf die folgende Struktur:

```
0           EKD@JO63rx_Dambeck.RSpectro
0-0             1073217600
0-0-0             FTLight
0-0-1             2004-01-12
0-1             Antenne
0-1-0             Parabolspiegel 90cm
```

**Umgang mit Änderungen**

Adressen können von besonderem Vorteil sein, wenn vorhandene Strukturen geändert werden müssen ohne dass Programme, welche sich auf die alten Strukturen verlassen in Mitleidenschaft gezogen werden sollen, wie z.B.:

Altes Format:

```
Frequenz:GHz,10.600
```

Neues Format:

```
Frequenz:GHz,10.600,Start,Schritt,Ende,Standard
   0-2,10.500
   0-3,0.00025
   0-4,12.750
   0-5,0-1
```

Die Struktur des neuen Formates würde folgendermaßen aussehen:

```
0       Frequenz
0-0        GHz
0-1        10.600
0-2        Start
0-2-0         10.500
0-3        Schritt
0-3-0         0.00025
0-4        Ende
0-4-0         12.750
0-5        Standard
0-5-0         10.600
```

Der letzte Wert (0-5-0) wäre der gleiche Wert wie (0-1) weil ein Link darauf verweist. Dies gestattet es mit der alten Struktur kompatibel zu bleiben, welche die Frequenz auf der Position (0-1) hatte. Alle Anwendungen, welche die Frequenz auf dieser Position erwarten, werden mit der neuen Datenstruktur auch weiterhin funktionsfähig bleiben. Zusätzlich können neue Anwendungen vorteilhaft die erweiterten Daten verwenden, welche neben der Frequenz auch eine Anfangsfrequenz, einen Frequenzschritt sowie eine Endfrequenz bereitstellen.

Somit zeigt das vorherige Beispiel eine zweite Methode für das Erzielen von Kompatibilität zwischen den Datenstrukturen, wobei die Definition der neuen Datenstruktur dafür sorgt, dass die Verbindung zu existierenden Anwendungen nicht unterbrochen wird. Zusammen mit der zuvor beschriebenen Verwendung von kompletten Pfadnamen für das Referenzieren von externen Daten, wo immer dies möglich ist, ergibt dies eine praktische Lösung für das Ändern von Datenstrukturen. Diese können nun über sehr lange Zeiträume und über mehrere Generationen von Anwendungsprogrammen angepasst, erweitert oder teilweise neu definiert werden, wann immer neue Anforderungen dies notwendig machen.

Die Möglichkeit zum Anpassen, Erweitern und teilweise neu Definieren von Datenstrukturen wird in der Radioastronomie, insbesondere wegen der sehr langen Zeiträume von Datenaufzeichnungen, für extrem wichtig erachtet. Eine Datenbank mit detaillierten Geschäftsdaten einer Firma wird nach einigen Jahrzehnten eventuell keinerlei Bedeutung mehr haben, weil es die Firma dann vielleicht schon lange nicht mehr gibt. Das Gegenteil ist bei radioastronomischen Daten der Fall. Das Vorhandensein von Jahrzehnte alten Daten von Radioquellen kann eine Schlüsselrolle spielen, wenn es um das Aufstellen von dynamischen Modellen der Radioquelle geht. Daher müssen die Daten über sehr lange Zeit in lesbarer Form erhalten bleiben. Die FTLight-Spezifikation ist diesem Ziel verpflichtet.

# Entropie-Modus

Der Entropie-Modus optimiert das FTLight-Protokoll für die folgenden zwei Anwendungsfälle:

- Nutzdatenübertragung mit hoher Geschwindigkeit bis nahezu 100% der Bandbreite
- Implementierung mit einfacher Logik z.B. für FPGA

In beiden Fällen  bleibt das Grundkonzept des FTLight-Protokolls bezüglich Unbeschränktheit sowohl bei der Größe von Informationselementen als auch bei der Tiefe der Informationshierarchie erhalten. Gegenüber anderen Modi gelten beim Entropie-Modus die folgenden Einschränkungen und Besonderheiten:

- der Entropie-Modus gilt für eine ganze Zeile und kann nicht mit anderen Elementen vermischt werden
- die Adressierung beginnt stets unterhalb des FTLI und es können daher nur Daten für diesen einen FTLI übertragen werden
- es werden nur bereits existierende Knoten der Informationshierarchie bedient
- es werden nur Knoten der jeweils untersten Ebene der Informationshierarchie adressiert sowie die Eltern-Knoten eines Ringpuffers auf unterster Ebene
- in einem Ringpuffer wird nach einmaliger Adressierung des Eltern-Knotens fortlaufend in die aufeinander folgenden Speicherplätze beginnend bei der aktuellen Position geschrieben
- falls die empfangenen Bits den letzten Speicherplatz nicht vollständig füllen, dann wird dieser mit „0“ aufgefüllt.
- falls im Entropie-Modus am Ende keine Byte-Grenze erreicht wird, dann werden die verbleibenden Bits beim Senden mit „0“-Füllbits aufgefüllt und beim Empfang verworfen
- wenn im Entropie-Modus Datenblöcke ohne Byte-Ausrichtung aufeinander folgen, dann können diese (gleichbedeutend) mit oder ohne „0“-Füllbits übertragen werden

*Übertragungsrahmen:*

Die Aktivierung des Entropie-Modus erfolgt durch ein gesetztes Bit („1“) am Anfang des Datenstroms. Bei allen anderen Modi ist das erste Bit stets rückgesetzt („0“).

Die Anzahl der gesetzten Bits am Anfang eines Datenstroms vor dem ersten „0“-Bit stellen einen Exponent zur Basis 2 für die Längenfestlegung eines Übertragungsrahmens dar, z.B.

„**1110**.....“

legt einen Übertragungsrahmen mit Byte-Ausrichtung (8 Bit) fest. Im Abstand dieses Rahmens zeigt das jeweils erste Bit eine Fortsetzung des Entropie-Modus an, falls es auf „1“ gesetzt ist. Andernfalls wird bei „0“ ein letzter Rahmen im Entropie-Modus angezeigt, z.B.

„**1110**xxxx**1**xxxxxxx**0**xxxxxxx“

überträgt mit einem Rahmen von 8 Bit insgesamt 3 Byte mit 18 Bit Nutzdaten.

Falls die Gesamtlänge aller Rahmen nicht auf eine Byte-Grenze fällt, dann müssen vor dem Verlassen des Entropie-Modus die verbleibenden Bits auf 0 gesetzt werden, z.B.

„**100**x“ oder „**110**x**1**xxx**0**xxx“ müssen mit jeweils vier „0“ Bits aufgefüllt werden zu:

„**100**x0000“ bzw. „**110**x**1**xxx**0**xxx0000“

Das Auffüllen mit Nullbits kann optional entfallen, wenn sich ein weiterer Datenblock im Entropie-Modus an einen Datenblock ohne Byte-Ausrichtung anschließt.

*Adressierung:*

Für die Adressierung auf jeder Ebene der Informationshierarchie werden N Bits verwendet. Die Anzahl N wird durch die Anzahl der 1-Bits beginnend beim ersten Bit der Nutzdaten nach den Bits zur Kennzeichnung der Rahmenlänge signalisiert. Diese Länge kann im speziellen Fall auch 0 sein. Es gibt keine Beschränkung zur maximalen Anzahl der Adressbits entsprechend dem Grundkonzept des FTLight-Protokolls zur Vermeidung von jeglichen Beschränkungen im Design.

Die Adresse wird Ebene für Ebene aus jeweils N aufeinanderfolgenden Bits gebildet ohne einen weiteren Trenner zwischen den Adressen der verschiedenen Ebenen. Die Adressierung endet, sobald ein letztes Element in der Informationshierarchie erreicht ist. Es wird in keinem Fall ein neues Element erzeugt. Alle weiteren Bits stellen stattdessen die Information für das angewählte Element dar, beginnend beim höchstwertigen Bit. Wenn weniger Bits übertragen wurden als wie das Element erwartet, dann werden die niederwertigen Bits mit 0 aufgefüllt.

Im Falle eines Ringpuffers gilt die Adresse des übergeordneten Elementes (Eltern-Element) als letztes zu adressierendes Element der Hierarchie. Die Information wird jedoch eine Ebene tiefer fortlaufend in die Speicherplätze des Ringpuffers eingetragen.

*Informationsbits:*

Die Anzahl der Informationsbits für das angewählte Element ist unbeschränkt. Je mehr Bits als Information übertragen werden, desto mehr nähert sich die Ausnutzung der Bandbreite dem theoretischen Maximum von 100% der zur Verfügung stehenden Bandbreite.

Die Adressierung bei den nachfolgenden Beispielen beginnt in jedem Fall unterhalb des FTLI in der darunter liegenden Ebene.

Das folgende Bitmuster ist ein Beispiel für ein Adressbit (**10**) „a“ und 8 Informationsbits „i“ bei 8-Bit Übertragungsrahmen mit einer Effizienz von 50% (8 Bit Nutzdaten in 16 übertragenen Bits).

„**1110 10** a i **0** i i i i i i i“

Wenn z.B 4 Adressbits (**11110**) erforderlich sind dann finden bei einem 8-Bit Übertragungsrahmen in 4 Byte bis zu 16 Informationsbits Platz, was ebenfalls einer Effizienz von 50% entspricht:

„**1110 1111 10** a a a a i i **1** i i i i i i i **0** i i i i i i i"

Wenn die Informationshierarchie unterhalb des FTLI einen Ringpuffer mit 8-Bit Speicherplätzen und ansonsten keine weiteren Elemente enthält, dann können z.B. 7 Byte Information in 8 Byte bei einer Rahmengröße von 32 Bit mit einer Effizienz von 87,5% übertragen werden:

„**111110 0** i i i i i i i i i i i i i i i i i i i i **0** i i i i i i i i i i i i i i i i i i i i"

Für eine weitere Erhöhung der Übertragungseffizienz muss die Größe der übertragenen Frames weiter erhöht werden. Wenn dabei wie im folgenden Beispiel 2046 Informationsbytes (16368 Bit) in 2048 Byte (16384 Bit) übertragen werden, dann erreicht die Effizienz bereits 99,9% der zur Verfügung stehenden Bandbreite:

„**11111111 111110 0** i i i i i i i i ... i i i i i i i **0** i i i i i i i i i i i i i i i ... i i i i i i i"

Wenn zusätzlich einige Adressbits benötigt werden, dann tritt bei großen Frames nur ein geringes Absinken der Effizienz auf, wie das folgende Beispiel zeigt. Hier werden in 1024 Bit insgesamt 1008 Informationsbits und zwei Adressbits übertragen, was einer Effizienz von 98,4% entspricht:

„**11111111 10 110** a a i i i i i i i i ... i i i i i i i **0** i i i i i i i i i i i i i i i ... i i i i i i i"

Die fehlenden 16 Bits bis zu einer "geraden" Anzahl von 1024 Nutzbits können mit kleinem Frame von 8 Bit ergänzt werden, wobei sich eine durchschnittliche Effizienz von 97,0% ergibt:

„**1110 110** a **1** a i i i i i i i **1** i i i i i i i **0** i i i 0 0 0 0"

# Framework- Funktionalität

Obwohl der Zweck einer Datensammlung im Speichern von Nutzdaten besteht, so ist es dennoch nützlich, Zusatzinformationen für die Beschreibung des Inhalts der einzelnen Felder zur Verfügung zu haben. Derartige Daten werden Metadaten genannt. Weil eine FTLight-Datei ihrerseits jedoch bereits viele Metadaten enthält, so könnte die Beschreibung dieser Metadaten als Meta-Metadaten bezeichnet werden. Stattdessen werden solche Daten im Kontext der FTLight-Spezifikation als Framework bezeichnet.

## Optionale Werte

Framework-Informationen werden durch leere Datenfelder eingeleitet, wie sie zum Beispiel durch einen zweifachen Doppelpunkt im Anschluss an eine Pfaddefinition entstehen. Alle Daten, welche dem zweifachen Doppelpunkt folgen, werden zu optionalen Werten, welche in das hierarchisch darüber liegende leere Feld eingetragen werden können.

Zum Beispiel:

```
Radioquelle::Sonne,Krabbennebel,3C353
```

Die Werte “Sonne”, “Krabbennebel” und 3C353 stellen die optionalen Werte dar, welche dem Feld “Radioquelle” als Wert auf der hierarchisch darunterliegenden Ebene zugewiesen werden können

## Kommentare

Ein Kommentar wird durch zwei benachbarte leere Datenfelder eingeleitet, wie zum Beispiel:

```
Radioquelle:::Dies ist der Name der beobachteten Radioquelle.
```

Schlüsselworte auf der Ebene von Kommentaren können dazu benutzt werden, um Empfehlungen für das Ausfüllen der Datenfelder zu geben, zum Beispiel:

```
Radioquelle:::Limit:Zeichen:80
```

Die Empfehlung für “Radioquelle” ist in diesem Fall, dass sie nicht mehr als 80 Zeichen lang sein sollte.

## Standardvorgaben

Sobald man ein Element am Ende einer Reihe von optionalen Werten anfügt wird dieses Element zur Standardvorgabe und die gesamte Hierarchie von darüber liegenden optionalen Werten wird an der Zielposition eingetragen. Diese Standardvorgabe kann später dadurch überschrieben werden, dass ein anderer optionaler Wert aus dem Wertevorrat als reales Element in die FTLight-Datei eingetragen wird, zum Beispiel:

```
Beobachtung::Radioquelle:Sonne
Beobachtung::Radioquelle::Sonne,Krabbennebel,3C353
...
Beobachtung:Radioquelle:Krabbennebel
```

Zunächst wird “Sonne” als Standardvorgabe für “Radioquelle” eingetragen, was später durch den Eintrag “Krabbennebel” überschrieben wird.

## Mehrfachspezifikationen

Im Falle von mehreren gleichförmigen Datenstrukturen ist es sinnvoll, die Framework-Daten nur einmal einheitlich für alle Datenstrukturen anzugeben. Bei der Arbeit mit mehreren gleichartigen Empfangskanälen kann zum Beispiel die Angabe für die Kanalnummer in der Pfadangabe freigelassen werden, wodurch sich die anschließenden Festlegungen auf alle Kanäle beziehen:

```
Empfangskanal,::Abtastrate:Hz
```

## Interpretation

Framework-Festlegungen werden in der Regel in einer speziellen Framework-Datei abgelegt, welche von den darauf aufbauenden Datendateien referenziert wird. Der Inhalt einer Framework-Datei ist als Hilfe zum Ausfüllen von Datenstrukturen gedacht, zum Beispiel durch das Vorgeben geeigneter Standardwerte sowie weiterer Optionen sowie auch für das Bereitstellen von Erläuterungen für das Ausfüllen der Felder. Die Framework-Daten bewirken niemals eine Beschränkung für das Ausfüllen von Datenstrukturen, jedoch werden sie Empfehlungen für sinnvolle Einschränkungen beim Ausfüllen vorgeben, wie z.B. die Anzahl der Zeichen in:

```
Radioquelle:::Limit:Zeichen:80
```

Dennoch ist es Sache des Benutzers von Framework-Daten, ob die Einschränkungen beachtet werden oder auch nicht.

# Datenintegrität

Das Ziel einer Unterstützung von Datenintegrität ist es zum einen, mögliche Datenverfälschungen zu erkennen und zum anderen ein Konzept anzubieten, mit dessen Hilfe soviel als möglich der noch unversehrten Daten eines defekten Files/Streams wiedergewonnen werden können.

Die Unterstützung von Datenintegrität in einem FTLight File/Stream basiert auf einer Zeile als kleinster Einheit, für die die Datenkonsistenz geprüft werden kann. Daher kann eine Prüfsumme (optional) an das Ende einer Zeile angehängt werden, welche von allen Zeichen der Zeile vor der Prüfsumme zuzüglich einer Zeilennummer, welche zeitweilig die Stelle der Prüfsumme einnimmt, berechnet wird.

Die Prüfsumme wird als Binärfeld eingesetzt, welches durch ein vorangehendes Gleichheitszeichen eingeleitet wird. Wenn ein Binärfeld am Ende einer Zeile nach einem Gleichheitszeichen erscheint, so handelt es sich um die Prüfsumme.

## Prüfsummenberechnung

Eine Prüfsumme wird durch ein oder mehrere Symbole des Binärdatenformats gebildet. Abhängig von der Anzahl der binären Symbole im letzten Feld der Zeile wird die Prüfsumme 216 zur Potenz der Anzahl sein, z.B. 216*216 = 46656 im Falle von zwei Symbolen oder 10077696 im Falle von drei Symbolen.

Die Anzahl der Symbole sollte in Abhängigkeit von der Länge der Zeile gewählt werden. Sie kann von einer Zeile zur nächsten variieren. Für eine kurze Textzeile wird ein einzelnes Symbol in der Regel ausreichend sein. Falls jedoch mehrere Gigabyte von Daten in einem einzelnen Element untergebracht werden, z.B. von einer Audio/Video-Datei oder wenn eine große FTLight-Datei als Binärdaten in einem Element gekapselt wird dann kann es empfehlenswert sein, entsprechend mehrere Symbole für die Darstellung der Prüfsumme zu verwenden. Es gibt vom Konzept her keine Begrenzung für die Anzahl der Symbole in einer Prüfsumme.

Nach Festlegung einer Symbolanzahl wird die Prüfsumme dadurch berechnet, dass der Rest beim Teilen der gesamten Zeile durch die Basis der Prüfsumme ermittelt wird. Vor Beginn dieser Rechnung wird die Zeilennummer (als ASCII-String) beginnend mit 1 für die erste Zeile an Stelle der Prüfsumme eingetragen.

Beispiel: Einstellige Prüfsumme für die Zeile 7

```
,Data=7
```

| Zeichen | Wert | Teiler Rest |
|---|---|---|
| , | 44 | 44 % 216 = 44 |
| D | 68 | ( 44*256+ 68) % 216 = 100 |
| a | 97 | (100*256+ 97) % 216 = 209 |
| t | 116 | (209*256+116) % 216 = 52 |
| a | 97 | ( 52*256+ 97) % 216 = 17 |
| = | 61 | ( 17*256+ 61) % 216 = 93 |
| 7 | 55 | ( 93*256+ 55) % 216 = **103** |

Das Zeichen welches dem Symbol 103 entspricht ist 103 + 32 = 135 entsprechend den zuvor für die Bildung des Binärdatentyps aufgestellten Regeln. Dieser Wert entspricht dem Zeichen ‘‡‘. Daher wird die Zeile mit der Prüfsumme wie folgt aussehen:

```
,Data=‡
```

Der Empfänger der Zeile wird die gleichen Rechnungen durchführen und wird dadurch in der Lage sein, die Konsistenz der Daten durch Vergleich mit der Prüfsumme am Ende der Zeile zu

überprüfen. Bevor die Rechnungen durchgeführt werden muss der Empfänger die Prüfsumme durch eine Zeilennummer ersetzen, welche Empfänger-seitig zu bilden ist. Daher wird die Zeilennummer ebenso überprüft, ohne dass diese explizit übermittelt werden muss.

## Wiederherstellung defekter Daten

Gelegentlich kann es vorkommen, dass ein einzelnes Byte oder eine Serie von Bytes in einem File/Stream verfälscht werden. Obwohl eine Prüfsumme das Feststellen einer Datenverfälschung ermöglicht, so kann sie dennoch keinen Beitrag zum Wiederherstellen der ursprünglichen Daten leisten. Daher wird es von den intakten Zeilen abhängen, ob Informationen wiederhergestellt werden können. Angenommen der Pfad der Information ist durch die defekte Zeile nicht verändert worden, so werden alle anderen Daten ihre Stellung in der Datenhierarchie beibehalten. Lediglich die defekte Zeile wird in diesem Fall verloren sein.

Falls jedoch in der defekten Zeile der Pfad geändert wurde und die darauffolgende Zeile sich auf den aktuellen Pfad bezieht dann werden alle nachfolgenden Informationen an eine falsche Stelle in der Informationshierarchie gesetzt werden. In diesem Fall ist der einzig mögliche Ausweg, die Datei in einem Texteditor zu öffnen und die defekte Zeile von Hand richtig zu stellen, soweit deren Inhalt bekannt ist oder aus der Struktur abgeleitet werden kann.

# FTLight-Archiv

Wenn ein Computer eine FTLight-Datei generiert oder einen FTLight-Stream von einem anderen Computer empfängt dann wird der Inhalt dieses Files/Streams in der Regel auf die Festplatte geschrieben. Der folgende Archivaufbau gestattet es, beliebige FTLight-Dateien auf die Festplatte zu schreiben, ohne dass Konflikte mit bereits dort vorhandenen früheren Dateien befürchtet werden müssen.

Der Schlüssel zum Erreichen eines eindeutigen Namens für den Pfad und die Datei ist der Zeitstempel für das Erstellen der Datei. Als eine unveränderliche Regel wird der Zeitstempel des Erstellens einer Datei immer auf Position “0-0” geschrieben und folgt damit direkt dem Identifikator auf der Position “0” in jedem FTLight File/Stream. Dies ist eine der fest definierten Eigenschaften von einem FTLight File/Stream, wo Funktionalität mit verbunden ist und aus diesem Grund muss diese Regel für alle Zeiten erhalten bleiben.

Die Verzeichnisstruktur in einem FTLight-Archiv ist wie folgt definiert:

1 – **FTLight**
2 – **Ortsbezeichner** (Galaxie-Zentralsonne-Sonnensystem-Planet_Locator_lokaler-Ortsname)
3 – **Identifikator** (Personenkennung@Ortsbezeichner.Gerätetechnik)
4 – **Jahr**
5 – **Monat**
6 – **Tag**
7 – **Stunde**
8 – **Minute**
9 – **Sekunde**
10 – **Millisekunde**
11 – …

Abhängig von der Zeitdauer, die eine FTLight-Datei umfasst, wird die eigentliche Datei zum Beispiel in einem ‘Sekunden’-Ordner abgelegt oder auf einer höheren oder niedrigeren Ebene.Die Namen der Dateien werden dabei wie folgt gebildet, wobei die Werte in Klammern zu ersetzen sind:

(Jahr)-(Monat)-(Tag)_utc(Stunde)h(Minute)m(Sekunde)s.(Millisekunde)_(Identifikator).csv

Unter der Voraussetzung, dass es für einen gegebenen Identifikator zu jedem Zeitabschnitt nur eine einzige Datei geben kann, wird der gewählte Dateiname universell eindeutig sein. Anstelle eines Identifikator kann auch ein Bezeichner für den Dateityp zum Einsatz kommen. Die Dateiendung .csv wird als Vorzugsvariante vorgeschlagen, wodurch die Datei bequem in Excel zum Anschauen geöffnet werden kann.

Beispiel: FTLight-Archiv mit Dateien einer 6-Minuten und einer Millisekunden Zeitdauer

```
FTLight
  JO63rx_Dambeck
    EKD@JO63rx_Dambeck.RSpectro
      2004
        Jan
          20th
            utc06
              06m
              12m
              18m
              24m
                2004-01-20_utc06h24m_EKD@JO63rx_Dambeck.RSpectro.csv
              31m
                47s
                  719ms
```

Ein weiterer Vorteil zusätzlich zur Eindeutigkeit der Dateinamen ist es, dass Informationen sehr leicht wiedergefunden und überprüft werden können. Weiterhin können Informationen in beliebigen Portionen entnommen und mittels Diskette, CD, DVD, Ftp-Datei oder einem beliebigen anderen Medium oder Übertragungskanal zu anderen Computern übermittelt werden.

Beispiel:

Das Übertragen der Beobachtungsdaten eines ganzen Tages von einer Station zu einer anderen Station erfordert lediglich das Komprimieren des Verzeichnisses vom entsprechenden Tag sowie das Übermitteln der entstehenden Datei zur Empfängerseite. Der Empfänger kann die erhaltene Datei dem eigenen FTLight-Archiv direkt hinzufügen ohne Gefahr zu laufen, dass vorhandene Daten dadurch überschrieben oder beeinträchtigt werden könnten.

# Versionsverfolgung

Informationen können zum Entstehungszeitpunkt falsch oder unvollständig sein oder sie können sich mit der Zeit ändern. Daher ist eine Methode erforderlich, welche das Korrigieren, Erweitern und Ändern von Informationen gestattet. Da sich der Erstellungszeitpunkt einer Information niemals ändert besteht eine zusätzliche Anforderung darin, auch unterschiedliche Versionen einer gegebenen FTLight-Datei in einem Archiv zu verwalten.

Das wesentliche Mittel für das Hinzufügen von Versionen zu bereits existierenden Dateien im FTLight-Archiv ist das Zuweisen eines neuen Zeitstempels auf die Nullposition in der dem aktuellen Zeitstempel untergeordneten Informationsmenge. Jede weitere Version wird das Gleiche tun und der Nullposition in der dem aktuellen Zeitstempel untergeordneten Informationsmenge einen neuen Zeitstempel zuweisen. Der Identifikator der Person, welche die Änderung vorgenommen hat wird ebenfalls der neuen Informationsmenge zugewiesen, welche bereits den Zeitstempel der Änderung auf der Nullposition eingetragen hat.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,1073217600
,Frequenz:GHz,10.600
,Bandbreite:kHz,250
```

Etwa zwei Wochen später wird die Frequenzinformation durch eine Person mit dem Identifikator kantz@wegalink.eu von 10.6 GHz auf 10550 kHz geändert:

```
EKD@JO63rx_Dambeck.RSpectro,1073217600
,Frequenz:GHz,10.600
,Bandbreite:kHz,250
0-0:1075123807,kantz@wegalink.eu,Frequenz:kHz,10550
```

Der zweite Informationsblock kann folgendermaßen interpretiert werden:

- Diese Information gehört zum Identifikator EKD@JO63rx_Dambeck.RSpectro
- Die ursprüngliche Information ist mit dem Zeitstempel 1073217600 verknüpft
- Eine geänderte Version wurde mit dem Zeitstempel 1075123807 erzeugt
- Der Identifikator der Person, welche die Änderung vornahm, ist kantz@wegalink.eu
- Die geänderte Version hat eine neue 'Frequenz: kHz, 10550'

Zusammenfassend kann festgestellt werden, dass eine neue Version von einer bereits existierenden Version dadurch erstellt wird, dass der Pfad des aktuellen Zeitstempels durch einen neuen Zeitstempel erweitert wird. Weiterhin wird der Identifikator der Person, welche die Änderung vornahm der neuen Informationsmenge hinzugefügt. Ferner werden alle geänderten Elemente der neuen Informationsmenge hinzugefügt, so als ob sie auf der Ebene des ersten Zeitstempels wären.

# Zeitsynchronisation

Das Anfordern eines Zeitstempels von einem anderen System erfolgt durch Übertragung des FTLI des anderen Systems (Empfänger) gefolgt vom eigenen FTLI (Absender).

```
Beispiel:

EKD@JO63rx_Dambeck.RSpectro,EKD@JN58ve_Poing.Lyra
```

Als Antwort wird ein Informationspaket erwartet, welches außer den beiden FTLIs in umgekehrter Reihenfolge und einem Zeitstempel mit der aktuellen lokalen Zeit des anderen Systems keine weiteren Angaben enthält. Dabei handelt es sich um ein TIME-Kommando. Dieses muss sofort wieder ohne Verzögerung in gleicher Weise mit der aktuellen lokalen Zeit beantwortet werden.

```
Beispiel:

EKD@JN58ve_Poing.Lyra,EKD@JO63rx_Dambeck.RSpectro,1607798473.123456789

EKD@JO63rx_Dambeck.RSpectro,EKD@JN58ve_Poing.Lyra,1607798473.234567890
```

Die Aufgabe von TIME-Kommandos ist die Übertragung der lokalen Zeit des Absenders zu einem Empfänger. Es obliegt im weiteren dem Empfänger, ob die übertragene Zeit als eigene lokale Zeit übernommen wird oder ob lediglich die Differenz zwischen den lokalen Zeiten des Absenders und des Empfängers mit einer Referenz zum FTLI des Absenders für die weitere Kommunikation mit diesem gespeichert wird. Insbesondere bei Kommunikation mit mehreren Seiten wird in der Regel nur eine Gegenseite als Zeitnormal dienen. Für die anderen Kommunikationspartner wird lediglich die Differenz zur eigenen lokalen Zeit mit einer Referenz zum FTLI der Gegenseite gespeichert.

Nach einer zweiseitigen TIME-Übertragung kann der Initiator die Differenz zwischen der lokalen Zeit der Gegenseite und der eigenen lokalen Zeit anhand der übertragenen Zeitstempel ermitteln. Für das Erzielen einer höheren Präzision bei der Zeitsynchronisation kann eine Serie von TIME-Kommandos verwendet werden. Mittels statistischer Methoden können aus den registrierten Zeiten für das Absenden und Empfangen der TIME-Übertragungen deren mittlere Laufzeit ermittelt und diese als Korrekturwert für die Zeitsynchronisation berücksichtigt werden.

# Anfrage/Antwort-Verfahren für historische Daten

Zunächst soll der Vorgang des Sendens einer Anfrage und das nachfolgende Empfangen einer Antwort beschrieben werden, wofür anschließend ein Verfahren definiert wird:

- Die anfragende Seite sendet eine Anforderung nach einer Untermenge von Daten, welche zu einem spezifizierten Identifikator gehören
- Eine antwortende Seite hat die angefragten Daten und übersendet diese an die anfragende Seite
- Die anfragende Seite ergänzt ihr FTLight-Archiv mit der erhaltenen Antwort
- Die anfragende Seite kann in schneller Folge Anfragen senden und Antworten erhalten
- Die Antwort wird in jedem Fall die Originaldatei mit unverändertem Zeitstempel oder eine Untermenge davon sein
- Es können mehrere Antworten zu ein und derselben Originaldatei im Ergebnis von unterschiedlichen Anfragen eintreffen
- Mehrere Originaldateien bzw. Untermengen davon können im Ergebnis einer einzigen Anfrage eintreffen

## Anfragestruktur

Das Anfragen von Informationen von anderen Informationshierarchien besteht aus allgemeinen Elementen, welche eine Anfrage identifizieren sowie dem Spezifizieren der jeweils nachgefragten Information. Die Detailinformationen einer Anfrage müssen zwischen den beiden Seiten abgestimmt sein, während die allgemeinen Anfrageparameter immer gleich bleiben. Eine Anfrage wird mit den FTLIs von Empfänger und Absender eingeleitet und muss beim Absenden auf der Position 0 unterhalb des Absenders in der Regel einen aktuellen Zeitstempel enthalten. Dieser ist die Grundlage für eine fortlaufende Einschätzung der Zeitsynchronisation zwischen beiden Seiten.

### Allgemeine Anfrageelemente

Das in der FTLight-Spezifikation festgelegte Anfrageelement (QUERY/EMPTY=96, Back-Apostroph) symbolisiert die „Leere“ welche eine Antwort provozieren soll, um die „Leere“ zu füllen. Es stellt somit das allgemeine Anfrageelement dar, sobald es allein in der Position 0 in einer beliebigen Informationsmenge auftaucht. In diesem Fall ist die anfragende Seite an der Übermittlung der vollständigen Informationsmenge interessiert, welche sich im FTLight-Archiv der antwortenden Seite an der entsprechenden Stelle befindet, einschließlich aller hierarchisch untergeordneten Informationsmengen. Der Identifikator der anfragenden Seite wird dabei in der Position 0 in der dem Anfrageelement untergeordneten Informationsmenge übertragen.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,`,EKD@JN58ve_Poing.Lyra,1607798473.123456789
```

Der Anfragesteller mit dem Identifikator EKD@JN58ve_Poing.Lyra möchte alle Informationen erhalten, welche sich unterhalb des Identifikators EKD@JO63rx_Dambeck.RSpectro befinden. Da es sich hierbei um eine riesige Datenmenge handeln könnte, wären einige Einschränkungen zur Zeitperiode angebracht, um die Größe der erwarteten Antwort einzuschränken. Jedoch gehört dies bereits zu den speziellen Elementen, welche beide Seiten miteinander vereinbaren müssen.

Ein allgemeines Element für das Begrenzen einer Anfrage ist das Hinzufügen von ausgewählten Elementen innerhalb der Informationsmenge, welche das Anfrageelement auf der ersten Position enthält. Dies begrenzt die Anfrage auf die Untermenge an Daten, welche sich unterhalb der angegebenen Elemente in der Informationshierarchie befindet.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro:`,Frequenz,Bandbreite
0-0,EKD@JN58ve_Poing.Lyra,1607798473.123456789
```

Im obigen Beispiel werden nur die Frequenz und die Bandbreite angefragt. Dies würde die große Menge an Messdaten ausschließen, welche zum Beispiel unterhalb des Datenelementes angeordnet sind. Es würden jedoch alle Dateien zurückgesendet, wo sich die Elemente Frequenz und Bandbreite auf der zweiten Ebene, genau unterhalb vom Identifikator befinden. Dies könnte ebenfalls noch eine große Datenmenge sein, so dass auch in diesem Fall noch eine Einschränkung bezüglich des Zeitabschnittes sinnvoll wäre.

Ein gerade in Übertragung befindlicher Antwort-Stream kann jederzeit durch die anfragende Seite gestoppt werden. Das allgemeine Element zum Stoppen einer Datenübertragung ist ein Anfrageelement (QUERY) an Stelle des anfragenden Identifikators. Dieser wird eine Ebene tiefer auf der Position 0 der dem zweiten Anfrageoperator untergeordneten Informationsmenge gesetzt.

Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,`,`,EKD@JN58ve_Poing.Lyra,1607798473.123456789
```

Das obige Beispiel stoppt den Datenstrom, der von der antwortenden Seite mit dem Identifikator EKD@JO63rx_Dambeck.RSpectro an die anfragende Seite mit dem Identifikator EKD@JN58ve_Poing.Lyra gesendet wird.

**Anforderung von Identifikatoren**

Ein einzelnes Back-Apostroph gefolgt vom Identifikator der anfragenden Stelle fordert eine Liste aller Identifikatoren an. Die Antwort besteht in diesem Fall aus einer Liste von Identifikatoren zusammen mit dem am weitesten zurückliegenden Zeitstempel für jeden Identifikator.

Beispiel:

```
`,EKD@JN58ve_Poing.Lyra,1607798473.123456789
```

Die Antwort auf diese Anfrage könnte wie folgt aussehen:

```
EKD@JO63rx_Dambeck.RSpectro,1073212000
```

Die Antwort informiert die anfragende Seite über die Verfügbarkeit von Daten zum Identifikator EKD@JO63rx_Dambeck.RSpectro beginnend ab dem Zeitpunkt, welcher durch den Zeitstempel 1073212000 in Sekunden nach dem 1. Januar 1970 in UTC angegeben ist.

**Zeiteinschränkungen**

Eine Zeiteinschränkung gehört zu den zusätzlichen Elementen des Anfrage/Antwort-Verfahrens, welche zwischen den beiden miteinander kommunizierenden Seiten vereinbart werden muss. Eine Struktur dafür könnte wie folgt aussehen:

```
EKD@JO63rx_Dambeck.RSpectro,`,EKD@JN58ve_Poing.Lyra,1607798473.123456789
0-0:Start:UTC,1073217400
:Ende:,1073217800
:Intervall:Sekunden,20
```

Im obigen Beispiel wird ein 400 Sekunden langer Zeitabschnitt angefordert und die Messwerte werden in einem Zeitintervall von 20 Sekunden erwartet. Solche Daten eignen sich zum Beispiel für eine Voransicht.

**Arrayabfragen**

Die Abfrage eines Arrays an einer adressierten Position kann durch eine Adressangabe weiter spezifiziert und dadurch eingeschränkt werden, zum Beispiel:

`EKD@JO63rx_Dambeck.Array,Data`3`,EKD@JN58ve_Poing.Lyra,1607798473.123456789`

schränkt die Abfrage auf das dritte Element (Untermenge) eines Arrays an der Position 'Data' ein.

# Abonnieren von aktuellen Daten

Nachfolgend wird ein Verfahren zum Abonnieren von aktuellen Daten beschrieben, wodurch neu entstehende Daten („Ereignisse“) automatisch unmittelbar an eine anfragende Seite übertragen werden:

- die anfragende Seite eröffnet eine Verbindung zu einer Informationshierarchie, von der während der Dauer der Verbindung Daten automatisch empfangen werden sollen
- die anfragende Seite spezifiziert ein Datenelement in der Informationshierarchie, bei dessen Änderung unter bestimmten Bedingungen eine Übertragung zur anfragenden Seite erfolgt
- die anfragende Seite legt fest, ob die Übertragung von geänderten Daten nur einmalig oder bei jeder Änderung erfolgen soll
- bei Anforderung aller Änderungen kann die Übertragungshäufigkeit auf einen fest vorgegebenen Zyklus eingeschränkt werden, wobei eine Übertragung des aktuellen Wertes im vorgegebenen Zyklus auch dann erfolgt, wenn keine Änderung vorliegt
- zum spezifizierten Datenelement können auch Unterelemente mit übertragen werden wenn dies beim Abonnieren so festgelegt wird
- eine Übertragung wird auch dann durchgeführt, wenn sich eines der Unterelemente geändert hat, falls beim Abonnieren Unterelemente mit angefordert wurden
- neben einem einzelnen Datenelemente und dessen Unterelementen können auch Datensätze abonniert werden, wie sie beim Synchronschreiben entstehen
- die anfragende Seite kann einen Startzeitpunkt festlegen, ab wann mit der einmaligen oder mehrmaligen Übertragung von geänderten Daten begonnen werden soll
- ein Abonnement für geänderte Daten endet wenn die anfragende Seite dieses aufhebt oder wenn die Verbindung zur Informationshierarchie von einer der beiden Seiten beendet wird.

**Datenelement spezifizieren**

Das Anfordern eines Abonnements für geänderte Daten erfolgt, wie in der FTLight-Spezifikation für Anfragen festgelegt, durch ein Anfrageelement (QUERY=96, Back-Apostroph) auf der Position des angeforderten Datenelementes. Die Information auf der Position 0 unterhalb des Anfrageelementes legt die Häufigkeit der Übertragung fest.

Falls unterhalb des Anfrageelementes keine Information vorhanden ist, dann erfolgt eine einmalige unmittelbare Übertragung des Wertes an die anfragende Seite. Wenn dagegen eine Zahl auf der genannten Position steht, dann legt diese die Anzahl der Sekunden eines Übertragungszyklus fest, wobei 0 das Übertragen jeder Änderung bewirkt, zum Beispiel

```
EKD@JO63rx_Dambeck.RSpectro,Frequenz,`
EKD@JO63rx_Dambeck.RSpectro,Frequenz,`,0
EKD@JO63rx_Dambeck.RSpectro,Frequenz,`,2.5
```

Die erste Zeile bewirkt eine einmalige Abfrage der aktuellen Frequenz. Die zweite Zeile bewirkt, dass alle Änderungen der Frequenz unmittelbar zurück gegeben werden. Die dritte Zeile gibt Änderungen in einem festen Raster von 2.5 Sekunden zurück, wobei eine Übertragung des Wertes in diesem Zeitraster auch erfolgt, wenn keine Änderungen der Frequenz erfolgt sind.

Bei Angabe eines Übertragungszyklus kann zusätzlich darunter auf der Position 0 ein Zeitpunkt für den Start der Übertragung in Sekunden angegeben werden. Falls die Zeitangabe in der Vergangenheit liegt dann wird die Angabe als Zeitraster behandelt, z.B. 1 bedeutet den Start der Übertragung zu Beginn der nächsten Sekunde. Eine 0 dagegen beginnt die Übertragung mit der

nächsten Änderung des Wertes, zum Beispiel wenn eine einmalige Übertragung angefordert wurde. Eine Zeitangabe in der Zukunft startet die Übertragung mit Erreichen der angegebenen Zeit.

```
EKD@JO63rx_Dambeck.RSpectro,Frequenz,`,,0
EKD@JO63rx_Dambeck.RSpectro,Frequenz,`,0,1373217800
EKD@JO63rx_Dambeck.RSpectro,Frequenz,`,2,1
```

Die erste Zeile bewirkt eine einmalige Übertragung der Frequenz bei deren nächsten Änderung. Die zweite Zeile bewirkt, dass ab dem angegebenen absoluten Zeitpunkt (Sekunden nach dem 1. Januar 1970 UTC) alle Änderungen übertragen werden. Mit der dritten Zeile wird eine Übertragung im 2-Sekunden-Zeitraster beginnend ab der nächsten vollen Sekunde gestartet.

Wenn statt eines einzelnen Datenelementes alle Unterelemente eines Parent-Elementes übertragen werden sollen dann wird dies durch einen Doppelpunkt ':' zum Eintragen des Anfrageelementes auf der Position 0 unterhalb des Parent-Elementes bewirkt, zum Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,Frequenz:`
EKD@JO63rx_Dambeck.RSpectro,Frequenz:`:0:1373217800
EKD@JO63rx_Dambeck.RSpectro,Frequenz:`:2:1
```

Mit der ersten Zeile wird wie zuvor ein Anfrageelement auf die Position eines Frequenzwertes gesetzt, jedoch werden bei der Übertragung auch alle Unterelemente des Parent-Elementes 'Frequenz' mit erfasst und bei jeder Änderung von einem der Unterelemente an die anfragende Seite übertragen. Zum Ergänzen des Anfrageelementes mit einem Zyklus und Startzeitpunkt müssen in diesem Fall ebenfalls Doppelpunkte ':' verwendet werden um jeweils die Position 0 zu erreichen.

Beim Synchronschreiben werden mehrere Datenelemente referenziert, welche gleichzeitig mit neuen Daten beschrieben werden. Das Abonnieren von Datensätzen erfolgt durch Eintragen des Anfrageelementes auf Position 0 des '@' Operators für das Synchronschreiben, zum Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro
Zeit,Flux,Temperatur
[Sekunden seit 1.1.1970],[Jy],[°C],@:`
```

Ergänzende Angaben zum Zyklus und zum Startzeitpunkt werden wie bei einem einzelnen Datenelement jeweils auf Position 0 unterhalb des Anfrageelementes ergänzt, zum Beispiel

```
0-3-0:`:2:1
```

Zum Beenden eines Abonnements wird an der Position 0 unterhalb des Anfrageelementes ein weiteres Anfrageelemente eingetragen, zum Beispiel:

```
EKD@JO63rx_Dambeck.RSpectro,Frequenz,`,`
```

```
0-3-0:`:`
```

Die Übertragung von Änderungen der Frequenz oder eines Datensatzes werden dadurch beendet, auch wenn die Verbindung zur Informationshierarchie noch weiter bestehen bleiben sollte.

# Laufzeitinformationen

- work in progress -

EKD@JN58nc_Türkenfeld.Algorithm,1549200792
,volatile,`

,volatile,`,#12345

## Algorithmen

EKD@JN58nc_Türkenfeld.Algorithm,1549200792,<log level>,<message>,<context>
,big numbers,`,0,<implementation>
„`,<#hashAlgorithm1>,<#hashClient1>
...
„`,<#hashAlgorithmN>,<#hashClientN>

Client1-Bereich
„`,<#hashClient1>,<#hashAlgorithm1>,<workspace 1>

ClientN-Bereich
„`,<#hashClientN>,<#hashAlgorithmN>,<workspace 1>

Kommandos (an #hash-Elemente):

- RUN
- WAIT
- GET
- SET
- STOP
- REMOVE

# Appendix A

## A.1 Beispiel für das Speichern von mehrstufigen Metadaten vor einem Interferometrie-Datenblock

```
MCL@JN76ec_Ljubljana.SIDI,1108598400:FTLight,2005-03-03
,global metadata tag1:item1,item2,...
,global metadata tag2:item1,item2,...
,global metadata tagN:item1,item2,...
,metadata for channel1,metadata tag1: item1, item2,...
,,metadata tag2:item1, item2,...
,,metadata tagN: item1, item2,...
,metadata for channel2,metadata tag1: item1,item2,...
,,metadata tag2: item1,item2,...
,,metadata tagN: item1, item2,...
,metadata for channelN,metadata tag1: item1, item2,...
,,metadata tag2: item1,item2,...
,,metadata tagN: item1, item2,...
,correlation values for baseline 1
:Time,Correlation
:1108598400.123,0.9876
:1108598400.124,0.9875
:1108598400.125,0.9877
...
,correlation values for baseline 2
:Time,Correlation
:1108598400.123,0.9836
:1108598400.124,0.9835
:1108598400.125,0.9837
...
,correlation values for baseline 3
:Time,Correlation
:1108598400.123,0.3476
:1108598400.124,0.3475
:1108598400.125,0.3477
...
```

## A.2 Beispiel für einen Mehrkanalempfänger mit wahlfreier Kanalselektion

Frequenzkanäle werden wahlfrei und in unregelmäßigen Zeitintervallen abgefragt. In diesem Fall muss die volle Frequenz- und Zeitinformation mit jedem der Messwerte abgespeichert werden. Dies könnte folgendermaßen für eine erste Basislinie und in ähnlicher Weise für andere Basislinien aufgebaut werden:

```
EKD@JN58ve_Poing.RSpectro,1108598400:FTLight,2005-03-03
,Daten,Basislinie1:[m],12.35
:Zeit,Frequenz,Signalstärke
:[Sekunden seit 1970-01-01],[GHz],[0..4095],@
:1109462400.111,10.610,2745
:1109462400.239,10.670,2745
:1109462400.377,10.655,2745
```

## A.3 Beispiel für einen Mehrkanalempfänger mit regelmäßiger Kanalselektion

Eine vordefinierte Frequenzliste wird in regelmäßigen Zeitabständen abgetastet. In diesem Fall ist es ausreichend, die Liste aller Frequenzen nur einmal anzugeben und jeweils die Startzeit für einen ganzen Block von Abtastwerten hinzuzufügen. Dies kann folgendermaßen erfolgen:

```
EKD@JN58ve_Poing.RSpectro,1108598400:FTLight,2005-03-03
,Daten,Basislinie1:[m],12.35
```

```
:Zeit,Freq1,Freq2,Freq3,Freq4,Freq5
:[Sekunden seit 1970-01-01],[GHz],[GHz],[GHz],[GHz],[GHz]
:gleiche Zeitintervalle,10.630,10.635,10.640,10.645,10.650,@
:1109462400.100,2736,2850,2473,2945,2791
:1109462500.100,2335,2457,2272,2628,2437
:1109462400.100,2533,2593,2311,2593,2692
```