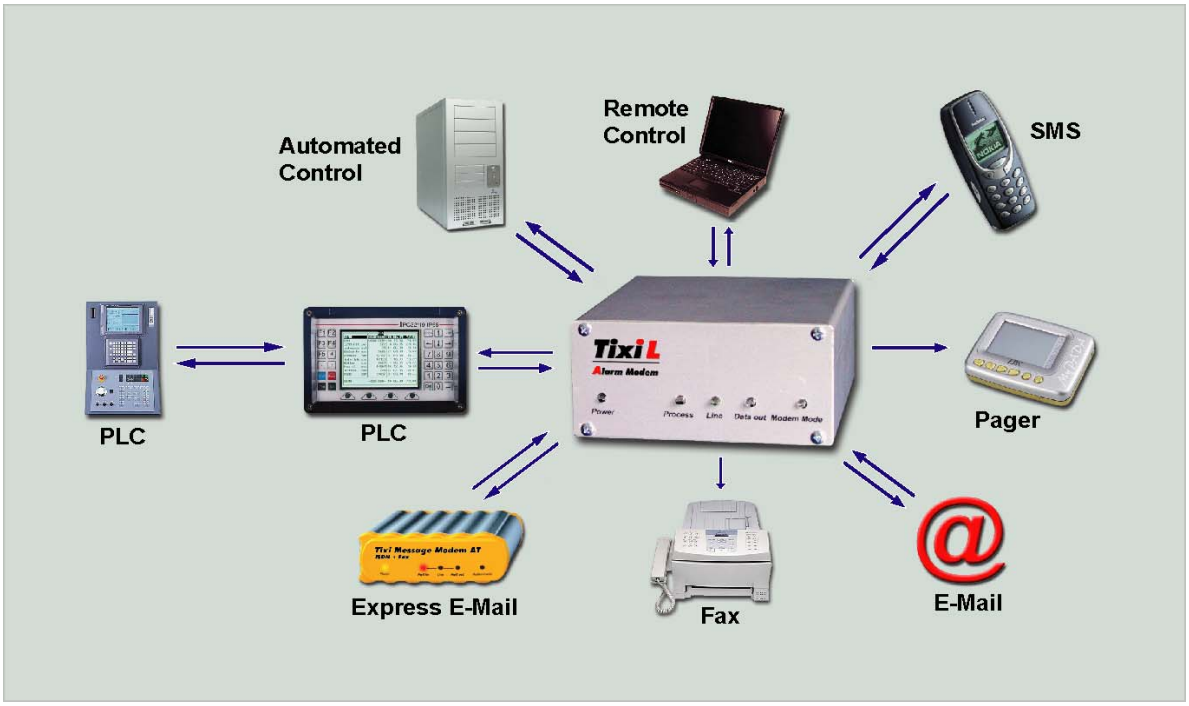
Tixi
Always online. Even offline.



TiXML-Tutorial

Für Firmware 2.2.12.0

Inhalt

1. Einführung

1.1. Syntaxerklärung XML - TiXML

1.1.1. Was ist XML ?

Im Folgenden soll ein lediglich kurzer Einblick in XML-Strukturen zur Darstellung von Daten gegeben werden. (Weitergehende Informationen zu Stylesheets u.a. findet man bei www.XML.org oder als kurzer Überblick bei Hubert Partl[1].)

Start- und End-Tags **für Blöcke von Daten**

„Die meisten Befehle (Tags) in XML-Anwendungen - wie auch in HTML - treten paarweise als Start- und End-Tags auf und geben an, welche Bedeutung der dazwischen liegende (eventuell durch weitere Tags unterteilte) Text hat.“ [1]

Bei den Tag-Namen ist die Groß- und Kleinschreibung zu beachten. Parameter zu den Tags werden als Attribute bezeichnet. Die Werte der Attribute werden in Anführungszeichen (“) oder Apostroph (‘ = ASCII-Zeichen: Single Quote) eingeschlossen (z.B. Autor =“Meyer“).

```
<Buch>  ... Text ...  </Buch>
```

oder

```
<Buch  Autor="Meyer"  ... >  ... Text ...  </Buch>
```

Ein kleines Adress-Datenbank-Beispiel soll das XML-Prinzip aufzeigen:

```
<person id="p4681">
  <vorname>Hubert</vorname>
  <zuname>Partl</zuname>
  <titel>Dr.</titel>
  <strasse>Manngasse 118</strasse>
  <plz>A-1220</plz>
  <ort>Wien</ort>
  <geburtstag>
    <tag>28</tag>
    <monat>März</monat>
    <jahr>1949</jahr>
  </geburtstag>
</person>
```

Da Tags geschachtelt werden können, lassen sich auch komplizierte Datenstrukturen übersichtlich darstellen.

Einzelne Elemente ohne End-Tag

Einen Spezialfall stellt ein Element ohne End-Tag dar. Es hat folgende Form:

```
<Buch />
```

oder

```
<Buch Autor="Meyer" ... />
```

Hier braucht der Tag-Name am Ende nicht wiederholt zu werden, was aus Platzgründen für kleine CPUs mit wenig Speicher ein wesentlicher Vorteil ist.

[1] Hans Hubert Partl, Wien, siehe auch http://www.boku.ac.at/htmleinf/xmlkurz.html

1.1.2. Unterschied zu TiXML ?

Tixi.Com hat für die Konfiguration der Alarm-Modems einen XML-Trick eingeführt und damit zwei Fliegen mit einer Klappe geschlagen: das Datenvolumen verringerte sich um 30 – 40 % und die Lesbarkeit der XML-Datei verbesserte sich erheblich, denn es stehen keine doppelten Informationen mehr in einer einzelnen Zeile.

Normale XML-Darstellung:

```
<vorname>Hubert</vorname>
```

TiXML-Darstellung:

```
<vorname _= "Hubert"/>
```

Das in XML zugelassene Zeichen „_“ wird hier als Dummy-Attribut für den Tag benutzt, dem als Variable ein Wert zugewiesen werden soll.

Das Beispiel von oben sieht also in TiXML so aus:

```
<person id="p4681" >
  <vorname _="Hubert" />
  <zuname _="Partl" />
  <Titel _="Dr. " />
  <strasse _="Manngasse 118" />
  <plz _="A-1220" />
  <ort _="Wien" />
  <geburtstag>
    <tag _="28"/>
    <monat _="März"/>
    <jahr _="1949"/>
  </geburtstag>
</person>
```

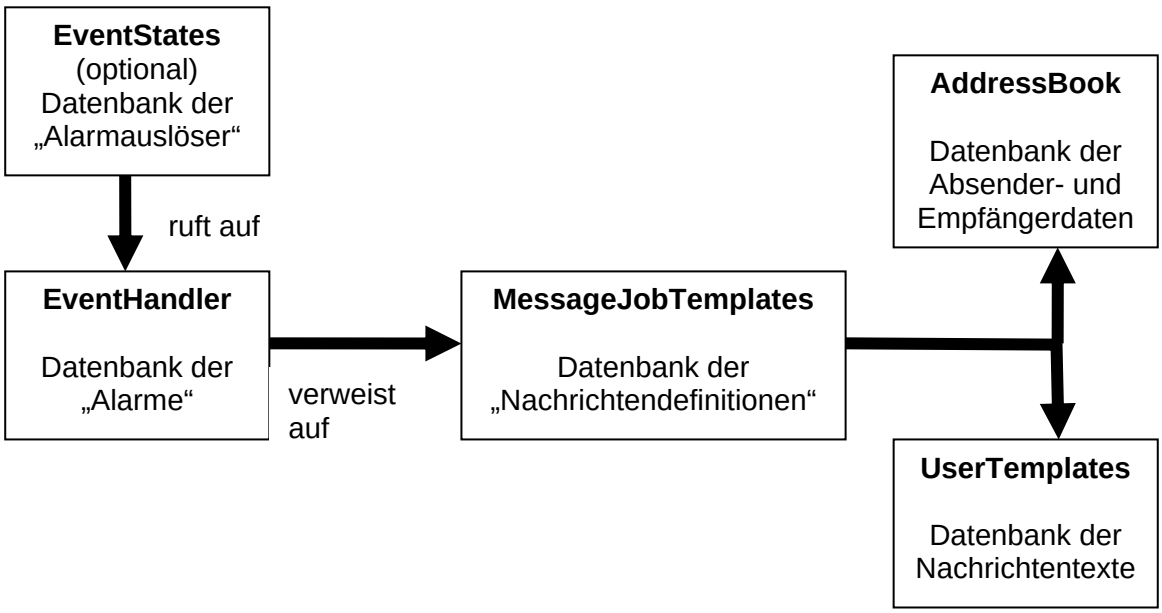
TiXML-Frames, d.h. TiXML-Datensätze oder Befehle, müssen beim Übertragen an das Alarm Modem in eckige Klammern gefasst werden. Dadurch erkannt das Modem, das es sich um Daten für den TiXML-Processor handelt, z.B.:

```
[<DoOn _=”Feuer”/>]
```

1.2. Überblick Zusammenhänge

Um den Einstieg in die TiXML-Parametrierung zu erleichtern, werden in diesem Kapitel anhand einer Störmeldung die Zusammenhänge der einzelnen TiXML-Datenbanken erklärt.

Ein im Tixi Alarm Modem hinterlegter einfacher Alarm besteht im Wesentlichen aus Einträgen in fünf Datenbanken, die miteinander verkettet sind.



In diesem Beispiel werden die Zusammenhänge eines Alarms „Feuer“, das als Faxmeldung an den Empfänger „Feuerwehr“ verschickt werden soll, erklärt.

An erster Stelle der Prozessabarbeitung steht der „**EventHandler**“:

```
[<SetConfig _="EVENTS" ver="y">
     <EventHandler>
          …
          <Feuer>
               <SendMail _="MessageJobTemplates/Feuer" value="0"/>
          </Feuer>
          …
     </EventHandler>
</SetConfig>]
```

Dieser verweist mit der Anweisung „SendMail“ auf die Datenbank mit den **MessageJobTemplates**:

```
[<SetConfig _="TEMPLATE" ver="y">
      <MessageJobTemplates>
            …
            <Feuer _="TextFax">
               <Recipient _="/D/AddressBook/Feuerwehr"/>
               <Sender _="/D/AddressBook/MySelf"/>
               <Body _="/D/UserTemplates/Feuer"/>
                <Subject _="&#xae;/D/UserTemplates/S-Feuer/Subject;"/>
            </Feuer>
            …
      </MessageJobTemplates>
</SetConfig>]
```

Das MessageJobTemplate verweist wiederum auf das **Adressbuch** (AddressBook) mit dem Absender „MySelf“ und Empfänger „Feuerwehr“, sowie auf den zu verschickenden **Nachrichtentext** (UserTemplate) „Feuer“:

```
[<SetConfig _="TEMPLATE" ver="y">
      <AddressBook>
            …
            <MySelf hidden="1">
                  <Fax _="+49-30-24037497"/>
                  <SMS_No _="+49-30-24037497"/>
            </MySelf>
            <Feuerwehr>
                  <Fax _="+49-30-10"/>
                  <SMS_No _="+49-176-24434569"/>
                  <SMS_Provider _="AnnyWay"/>
            </Feuerwehr>
            ...
</AddressBook>
</SetConfig>]

[<SetConfig _="TEMPLATE" ver="y">
<UserTemplates>
<Feuer>
      <Subject _="Feueralarm ! "/>
      <Body>
            <E _="Die Scheune brennt !"/>
      </Body>
      </Feuer>
</UserTemplates>
</SetConfig>]
```

Um den Alarm auszulösen, müsste das Alarm Modem über die serielle Schnittstelle den Befehl [<DoOn _=”Feuer”/>] erhalten. Der Parameter „Feuer“ ist der Name des auszulösenden EventHandlers.

Man kann den Alarm allerdings auch aufgrund der Änderung einer Variable auslösen lassen, z.B. über einen digitalen Eingang „P0“. Dazu wird eine weitere Datenbank „**EventStates**“ herangezogen:

```
[<SetConfig _="PROCCFG" ver="y">
     <EventStates>
          <Feuer>
               <Enabled _="TRUE"/>
               <ProcessVar _="/Process/MB/IO/I/P0" flank="high"/>
               <Event _="Feuer"/>
          </Feuer>
     </EventStates>
</SetConfig>]
```

Der EventState ruft über den Eintrag <Event _=”Feuer”/> den am Anfang konfigurierten EventHandler „Feuer“ auf. Das EventState steht also an erster Stelle der Prozessabarbeitung.

Ähnliche Zusammenhänge gibt es auch für alle weiteren Aktionen, die das Alarm Modem verarbeiten kann. Im EventHandler werden dazu anstelle des „SendMail“-Befehls, der für das Versenden von Nachrichten steht, die Befehle für Datenlogging („Log“, „BinLog“) , Schalten („Set“), Alarmkaskaden („OnError“, „OnOK“) usw. eingefügt.

Die Verwendung der einzelnen Befehle wird in diesem TiXML-Tutorial erläutert.
Im Handbuch finden Sie Verweise auf Kapitel im TiXML-Reference Manual. Dort können Sie detailliert alle Parameter und Einstellungsmöglichkeiten für die einzelnen TiXML-Funktionen nachschlagen.

1.3. Vorgehensweise bei der Projekterstellung

Bevor Sie sich tiefer mit dem Tutorial beschäftigen, sollten Sie zunächst einmal klären, ob die Version des Handbuchs sowie der TICO-TiXML-Konsole zu der auf dem Alarm Modem verwendeten Software (Firmware) passt, da es historisch bedingt zwei unterschiedliche Dokumentationen gibt.

Die Alarm Modem Firmware bis Version 1.72.14 ist auf die Software TILA1 und eine ältere TICO ausgerichtet und wird in diesem Handbuch nicht behandelt (siehe Handbuch Code TUT-DE).

Die Alarm Modem Firmware ab Version 2.0.0.0 ist auf die Software TILA2 und die „TICO für FW 2.0“ (siehe Start-Menü-Name) ausgerichtet und Hauptbestandteil dieses Handbuchs. Einige der hier beschriebenen Funktionen erfordern allerdings die Firmware Version 2.2.12.0. Eine vollständige Kompatibilitätsliste können Sie dem TiXML-Reference Handbuch im Kapitel „Firmware“ entnehmen.

Sollten Sie eine ältere Firmware Version als die 2.2.12.0 verwenden, so raten im Fall von Problemen zum Update auf FW 2.2.12.0.

Vor dem eigentlichen Erstellen von TiXML-Projekten sollte zunächst ein Basisprojekt mit Hilfe von TILA2 erstellt werden. Dieses Basisprojekt sollte bereits alle Modem- und Geräteeinstellungen, die Absender- und Empfängerdaten, Nachrichtentexte, einfache Alarme, Schaltbefehle sowie ggf. SPS-Variablen enthalten.

Dieses Basisprojekt kann dann mit der TiXML-Konsole importiert und weiterverarbeitet werden. Dadurch erspart man sich den größten Teil der TiXML-Parametrierung, und braucht das Projekt nur noch um wenige Zeilen erweitern.

Der Vollständigkeit halber wird im Kapitel 2 der projektrelevante TiXML-Inhalt des Basisprojektes erläutert. Da diese Werte aber komfortabler über TILA2 zu parametrieren sind, kann dieses Kapitel auch übersprungen werden.

Erfahrene Anwender können TICO-Projekte aber auch komplett per Mausklick aus dem Template-Fenster zusammenstellen. Alle unterstützten Datenbanken und Elemente stehen dort zur Auswahl.

1.4. Info zu den begleitenden Projektbeispielen

Dem TiXML-Tutorial liegen TICO-Projektbeispiele zur praktischen Veranschaulichung der TiXML-Funktionen im Verzeichnis „Examples“ bei.
Das in TILA2 erstellte „Basis“-Projekt dient dabei als allgemeine Vorlage.

Es gibt zu jeder Funktion ein eigenständiges Beispielprojekt, welches Sie nach Ihren Bedürfnissen anpassen können.

Passen Sie zunächst das Projekt „basis.cnf“ Ihren Gegebenheiten (Telefonanschluß, Adressbuch) an, und verteilen Sie diese Grundeinstellungen über den Startmenüeintrag „Basiseinstellungen verteilen“ der „TICO für FW 2.0“ Programmgruppe.

Die Projekte sind für HM Alarm Modems geschrieben, können aber auch mit anderen Alarm Modems verwendet werden. Beachten Sie, dass Sie beim Test mit GSM-Modems (HG) in einigen Beispielen Einträge verändern müssen:

- Im Adressbuch ist als SMS-Provider „GSM“ anzugeben
- Im MessageJobTemplate ist „GSMSMS“ statt „SMS“ anzugeben.

1.5. Projektaustausch zwischen TILA2 und TICO

TILA2 und TICO sind derzeit nur in eine Richtung 100% kompatibel: Ein mit TILA2 erstelltes Projekt kann problemlos mit TICO ausgelesen werden.
Ein mit TICO erweitertes Projekt sollte nachfolgend nicht mehr mit TILA2 verändert werden. TILA2 bietet zwar im Menü „Datei“ die Möglichkeit an, TICO Projekte zu importieren, verwirft aber beim Bearbeiten alle Ergänzungen die es nicht kennt (d.h. interpretieren kann).

Zur komfortablen Änderung von Projektteilen durch den Endanwender, z.B. dem Adressbuch, bietet Tixi.Com die Software S-TILA an, in der alle Elemente ausgeblendet werden, die der Endanwender „zerstören“ könnte.

Um ein mit TILA2 erstelltes Projekt in TICO darzustellen, muss der Umweg über das Alarm Modem gewählt werden. Dazu überträgt man das Projekt mittels TILA2 in das Modem (Menüpunkt: Modem: Projekt auf Modem speichern), und ließt es anschließend in TICO wieder ein (Menüpunkt: Projekt – Open from TAM).

Wenn Sie bereits mit der TILA1 oder dem alten TiXML-Tutorial (für Firmware bis 1.72.14.0) gearbeitet haben, werden Sie feststellen, dass die TiXML-Tags in TILA2 mit festen Bezeichnern (z.B. Contact_0, Switch_0 usw.) statt der selbst gewählten Namen definiert werden (so wie es in Kapitel 1.2 noch zu sehen ist). Dies macht die Projekte zunächst etwas unübersichtlich, war aber für die Flexibilität der TILA2-Module unverzichtbar. Der Anwender bekommt aber über das neu hinzugefügte Attribut „name“ den selbst gewählten Namen angezeigt. Die Firmware wertet dieses Attribut nicht aus.

TILA2 fügt an einigen Stellen sowie auf dem Register “30-TILA2“ zusätzliche XML-Parameter zur Organisation der Daten ein. Diese sind bei der Programmierung über TICO nicht relevant, werden daher im Tutorial kursiv dargestellt, und nicht näher erläutert.

2. Die TiXML-Datenbanken

Die Software TICO stellt bei einem geöffneten Projekt jede TiXML-Datenbank auf einer eigenen Seite dar, die über die ProjectView Liste ausgewählt werden kann. Diese lassen sich auch übers Menü bzw. Kontextmenü entfernen und hinzufügen.

```
 1 – Location
 2 – UserData
 3 – AddressBook
 4 – ISP
5/6 – AccRights
7/8 – LogDefinition
 9 – EventLogging
10 – EventHandler
11 – MessageJobTemplates
12 – MessageText
13 – ProcessVars
14 - EventStates
15 – Schedule Conditions
16 – Schedule
17 – External
18 – SMS-Provider
19 – Incoming SMS-Provider
20 - Incoming CallerID Alarms
21 - Sequencer
…
30 – TILA2
```

1 – Location
Hier stehen die Standortrelevanten Daten wie z.B. die Land, Telefonnummer und Amtsholung

2 – UserData
Hier sind die Einstellungen zur Modemkommunikation hinterlegt, z.B. Wahlwiederholungen, ISDN MSN, Faxkennung und Rufannahme.

3 – AddressBook
Die Kontaktadressen des Absenders und der Empfänger, z.B. E-Mail Adresse, Fax- oder SMS-Nummern.

4 – ISP
Die Daten des Internetzugangs, z.B. Rufnummer, Anmeldung und E-Mail-Konto sowie der Internetzeitsynchronisierung.

5/6 - AccRights
Hier kann der Zugriffschutz festgelegt werden, durch welchen das Gerät lokal und bei Ferneinwahl sowie beim Fernwirken per SMS, Email oder Express-E-Mail geschützt ist.

7/8 – LogDefinition
Die Namen und Größen der verwendeten Logfiles für System- und Datenlogging sowie die Struktur der Datensätze beim Binärlogging werden hier angegeben.

9 – EventLogging
Hier wird die Art und der Umfang des Systemprotokolls definiert, z.B. Loggen von Fehlern bei der Alarmabarbeitung.

10 – EventHandler
Die Namen der Aktionen bzw. Alarme und die damit verknüpften Befehle, z.B. Nachricht senden oder Daten loggen.

11 – MessageJobTemplates
Hier werden Nachrichtentexte und Adreßbuchkontakte zu Alarmmeldungen verknüpft sowie deren Typ, z.B. SMS oder TextFax festgelegt.

12 – MessageText
Hier stehen die Nachrichtentexte und deren Betreffzeilen.

13 – ProcessVars
Hier können Variablen definiert und logische Operationen ausgeführt werden.

14 – EventStates
Die Bedingungen für eine Aktion werden hier beschrieben, z.B. bei welcher Variablenänderung ein bestimmter Alarm ausgelöst werden soll.

15 – Schedule Conditions
Hier können weitere Bedingungen für den Kalender angelegt werden, z.B. Ferienzeiten in denen eine Zeitschaltfunktion nicht ausgeführt werden soll.

16 – Schedule
Der Kalender des Alarm Modems wird hier parametriert, d.h. es können Zeitpunkte für Aktionen (z.B. Logging) angegeben werden (Zeitschaltuhr).

17 – External
Bei Verwendung einer SPS werden hier die Tabellen ihrer Variablen sowie die Einstellungen zur Kommunikation mit der SPS vorgenommen.

18 – SMS-Provider
Die Gateways für SMS-Versand werden hier mit den Einwahlrufummern und Protokollen hinterlegt.

19 – IncomingSMSProviders
Die Rufnummern (Service Center) für den Empfang von Festnetz-SMS werden hier angegeben.

20 – Incoming CallerID Alarms
Hier können Rufnummern bestimmte Aktionen zugewiesen werden. Bei einem Anruf dieser Nummer kann z.B. eine Garage geöffnet werden.

21 – Sequencer
Über den Sequencer können zu bestimmten Zeiten bestimmte Werte gesetzt werden.

22-29 – Erweiterte Register
Die hier angebotenen Einstellungen sind Spezialanwendungen, die in diesem Tutorial nicht verwendet werden. Die einzelnen Objekte sind in den TiXML-Reference-Handbüchern erläutert.

30 – TILA2
Auf diesem Register organisiert TILA2 die konfigurierten Elemente. Es lassen sich z.B. in TICO manipulierte Alarme vor TILA2 verstecken. Die Funktion ist derzeit noch nicht dokumentiert und auch nicht Bestandteil dieses Tutorials.

3. Die Basiskonfiguration

Die Basiskonfiguration (`basis.cnf`) stellt die grundlegenden Daten wie z.B. Standortparameter, Absender- Empfängeradressen oder den Internetzugang zusammen. Diese Daten können alle über TILA2 parametriert werden.

Die Nachfolgenden Parameter sind projektspezifisch und sollten Ihre persönlichen Gerätedaten enthalten.

3.1. Location (TiXML Reference Kapitel 3.3)

In der Location (Register `1-Location`) muß die Rufnummer des Tixi Alarm Modem geändert werden. Es ist die Nummer des Anschlusses einzutragen, an dem das Modem angeschlossen ist (ohne Landes und Ortskennzahl):

```
<PhoneNumber _="12345678"/>
```

Sollte die Ortskennzahl nicht die “30“ (Berlin) sein, so ist diese hier ebenfalls zu ändern (immer ohne führende Null, welche in Dt. die nationale Vorwahl darstellt):

```
<AreaCode _="30"/>
```

Wenn das Tixi Alarm Modem über eine Telefonanlage angeschlossen ist, kann zudem eine evtl. benötigte Amtsholung definiert werden. Bei einer “0“ wäre dies z.B.:

```
<LocalDialPrefix _="0"/>
<LongDialPrefix _="0"/>
```

3.2. UserData (TiXML Reference Kapitel 3.2)

Im Register “`2-UserData`“ muss der Name und die Kennung des Tixi Alarm Modem sowie ggf. die ISDN-Mehrgeräterufnummer eingetragen werden:

```
<BoxName _="Tixi Alarm Modem"/>
```

wird z.B. bei Faxen in der Kopfzeile als verwendet.

```
<BoxNumber _="+49-30-1234567"/>
```

wird in der Kopfzeile von Faxen als Kennung verwendet.

```
<IsdnDataChannelID _="*"/>
```

Legt die MSN vom Tixi Alarm Modem am ISDN-Anschluß fest. Diese ist wichtig, wenn zum Tixi Alarm Modem auch eine Remote-Verbindung hergestellt werden soll. Ein “*“ bedeutet “Rufe für jede Nummer annehmen“.

Der „Ring Counter“ regelt die Anzahl der Klingelzeichen bis zur Rufannahme. Bei „0“ ist die Rufannahme abgeschaltet. Ein anderer Wert entspricht der Anzahl der abzuwartenden Klingelzeichen, z.B. Annahme nach einem Klingelzeichen:

```
<RingCounter _="1"/>
```

PIN der SIM-Karte, falls nötig:

```
<Pin1 _="1234"/>
```

Achtung: Diese Einstellungen werden erst nach einem Neustart des Modems wirksam!

3.3. AddressBook (TiXML Reference Kapitel 3.4)

Im Adressbuch (Register `3-AddressBook`) werden die eigenen Absenderdaten `<MySelf>` sowie die Empfänger `<Contact_X>` definiert.
Für jeden verwendeten Nachrichtentyp (außer CityRuf) ist zwingend eine entsprechende Absenderadresse bzw. -nummer im Kontakt „MySelf“ nötig.

Die E-Mail Adresse, z.B.:
```
<Email _="Tixi-Training1@gmx.net"/>
```

Die SMS-Nummer , z.B.:
```
<SMS_No _="+49-172-1234567"/>
```
und der dazugehörige SMS-Provider (nur beim Empfänger), z.B.:
```
<SMS_Provider _="AnnyWay"/>
```
Mögliche Werte: `D1,D1_ISDN,D2,D2_ISDN,Eplus,Telekom,AnnyWay,…`

Die Faxnummer, z.B.:
```
<Fax _="+49-30-1234567"/>
```

Die Express-Mail Adresse, z.B.:
```
<Express-Email _="INFO+49-30-1234567"/>
```

Die CityRuf Nummer (nur beim Empfänger),z.B.:
```
<CityRuf _="3949000"/>
```
Und der dazugehörige Pager-Provider:
```
<Pager_Provider _="CityRuf"/>
```

Es können beliebig viele Empfänger mit beliebigen Namen hinzugefügt werden. Jeder Empfänger kann mehrere Adresstypen enthalten, z.B.:

```
<Contact_0>
     <INet _="info@tixi.com"/>
     <Fax _="+49-30-40608400"/>
     <Express-Email _="INFO+49-30-40608555"/>
     <SMS_No _="03040608300"/>
     <SMS_Provider _="AnnyWay"/>
     <CityRuf _="3949000"/>
     <Pager_Provider _="CityRuf"/>
</Contact_0>
```

Der in den Beispielen vorhandene Kontakt „OA“ wird später für automatische Rückantworten beim Fernwirken verwendet.

3.4. ISP (TiXML-Reference Kapitel 3.5)

Der Providerabschnitt (Register `4-ISP`) ist für einen Internet-by-Call Testaccount vordefiniert und sollte nach Möglichkeit durch einen eigenen Internetzugang ersetzt werden.

Wichtig sind:
Die Benutzerdaten für die Einwahl:
```
<PPPUserName _="Freenet"/>
<PPPPassword _="Freenet"/>
```

Der Server für ausgehende (Abschnitt SMTP) bzw. eingehende (Abschnitt POP3) Mails:
```
<mailserver_name _="mx.freenet.de"/>
```

Die POP3-Benutzerdaten des Mailservers:
```
<Username _="TIM-Test"/>
<Password _="xyz"/>
```

Bei Bedarf kann das Verfahren POP-vor-SMTP oder ESMTP im Abschnitt „SMTP“ aktiviert werden. Bei ESMTP müssen Sie dort zusätzlich noch Benutzerdaten (wie bei POP3) angeben.:
`<Flags _="POPBeforeSMTP"/>` bzw. `<Flags _="ESMTP"/>`

Die Einwahlnummer:
```
<RemotePhoneNumber _="+49-1019-01929"/>
```

Das Löschen der abgeholten Mails auf dem Mailserver kann deaktiviert werden:
```
<Flags _="DontDelete"/>
```

3.5. Logfiles (TiXML Reference Kapitel 4)

Das Tixi Alarm Modem bietet von sich aus 5 Logfiles an die der Überwachung von Prozessen dienen. Diese müssen jedoch erst im Dateisystem angelegt, sowie die Protokollierung aktiviert werden.

Angelegt werden die Logfiles im Register `7/8-LogDefinition` (size=Größe in Byte):

```
<LogDefinition>
<LogFiles>
  <JobReport size="10240"/>
  <Event size="10240"/>
  <Login size="10240"/>
  <IncomingMessage size="10240"/>
  <FailedIncomingCall size="10240"/>
</LogFiles>
<LogDefinition>
```

Im Register `9-EventLogging` wird auch konfiguriert, welche Ergebnisse der Prozesse protokolliert werden (mode: erste Stelle: `e=errors`, `o=OKs`, `a=all`, zweite Stelle optional: `v=verbous`):

```
<EventLogging>
     <JobReport mode="av" file="Job"/>
     <Event mode="av" file="Event"/>
     <Login mode="av" file="Login"/>
     <IncomingMessage mode="av" file="Incoming"/>
     <FailedIncomingCall mode="av" file="failed"/>
</EventLogging>
```

Das Logfile “JobReport” enthält Informationen über die abgearbeiteten Prozesse.
Das Logfile “Event” listet alle ausgeführten Events auf.
Das Logfile “Login” protokolliert Anmeldungen (lokal und remote) am Gerät.
Das Logfile “IncomingMessage” enthält Informationen eingegangene Nachrichten (Fernwirken).
Das Logfile “FailedIncomingCall” enthält Informationen über Fehler beim Fernwirken.

3.6. Zugriffsschutz (TiXML-Reference Kapitel 3.11)

Die Konfiguration sowie die Fernkonfiguration lassen sich durch Benutzerdaten schützen. Diese müssen bei der Einwahl oder vor jeder Konfiguration übermittelt werden.
Die Benutzerdaten können im Register `5/6-AccRights` eingefügt werden:

Der Schutz erfolgt durch ein Passwort (Benutzername optional), z.B.:

```
<AccRights>
     <Groups>
          <Admin>
               <LocalLogin AccLevel="1"/>
               <RemoteLogin AccLevel="1"/>
          </Admin>
     </Groups>
     <User _="Plain">
          <Def_LocalLogin Plain="Sonne" Group="Admin"/>
          <Def_RemoteLogin Plain="Mond" Group="Admin"/>
     </User>
</AccRights>

<Login>
     <Techniker _=“Sonne“/>
</Login>
```

Der lokale Login Befehl sieht dann wie folgt aus:
`[<Login _="PAP" password="Sonne" ver="y"/>]`

Sowie der remote Login Befehl:
`[<Login _="PAP" password="Mond" ver="y"/>]`

4. Konfiguration von Alarmfunktionen

Das Tixi Alarm Modem kann auf verschiedene Art zum Verschicken eines Alarms aufgefordert werden.

- per TiXML-Befehl an der RS232
- per SPS-Variable an der RS232 /422 /485
- per digitalen Eingang
- per eingehender Nachricht
- per Scheduler

Die folgenden Kapitel erläutern diese unterschiedlichen Alarmmöglichkeiten Schritt für Schritt auf.

4.1. Einfache Alarmfunktion

Zunächst soll eine einzelne Nachricht verschickt werden. Da diese Funktion auch in TILA2 vorhanden ist, kann dies dort sicher einfacher parametriert werden. Aus Gründen der Vollständigkeit wird hier trotzdem die Konfiguration in TiXML erklärt. Im Beispielprojekt `basis-messages.cnf` ist für jeden Nachrichtentyp ein eigener Alarm parametriert. Im Folgenden wird exemplarisch das Verschicken eines TextFax erläutert.

Wie im Kapitel 1.2 erwähnt, arbeitet das Alarm Modem bei einem Alarm Datenbanken in einer bestimmten Reihenfolge ab:
EventHandler – MessageJobTemplates – AddressBook – MessageText

4.1.1. Event Handler (TiXML Reference Kapitel 3.8)

Als erstes wird ein EventHandler erzeugt (Register `10-EventHandler`):

```
<Alarm_0 Name="Fax-Alarm">
     <SendMail _="MessageJobTemplates/Alarm_0"/>
</Alarm_0>
```

Der <Tag>-Name des EventHandlers (hier : `"Alarm_0"`) ist theoretisch frei wählbar (zwecks Einheitlichkeit möglichst TILA2-Syntax „`Alarm_X`“ beibehalten) und wird zum Auslösen des Alarms per `DoOn`-Befehl verwendet.

Der EventHandler enthält Befehle, die beim Eintreten dieses Events abgearbeitet werden. Bei diesem einfachen Alarm ist dies zunächst nur das verschicken einer Nachricht per „`SendMail`“.
Der Befehl `SendMail` verweist auf ein MessageJobTemplate “`Alarm_0`“.

4.1.2. MessageJobTemplate (TiXML Reference Kapitel 3.7)

Das MessageJobTemplate (Register `11-MessageJobTemplates`)enthält Angaben zum Nachrichtentyp, Absender/Empfänger sowie dem Nachrichteninhalt:

```
<Alarm_0 _="TextFax">
     <Recipient _="AddressBook/Contact_0"/>
     <Sender _="AddressBook/MySelf"/>
     <Subject _="&#xae;/D/UserTemplates/Message_3/Subject;"/>
     <Body _="/D/UserTemplates/Message_3/Body"/>
</Alarm_0>
```

Recipient und Sender sind mit einem Eintrag im zuvor erstellten Adressbuch verknüpft. Der Betreff (Subject) und der Nachrichtentext (Body) verweisen auf die UserTemplates Datenbank .

Zu beachten ist, daß bei SMS und CityRuf der `Body` weggelassen wird.

4.1.3. MessageText (TiXML Reference Kapitel 3.6)

Die MessageTexte werden in den UserTemplates (Register `12-MessageText`) abgelegt. Ein einfacher Text für den Fax-Alarm könnte z.B. so aussehen:

```
<Message_3 Name="Fax-Alarm" Type="Mail" UseSignature="0">
     <Subject _="This is the subject line of the Fax."/>
     <Body>
          <E _="This is the message body of the Fax."/>
          <E _="Enter as many lines as you need."/>
     </Body>
</Message_3>
```

Bei SMS und CityRuf wird nur der Betreff verwendet. Der Inhalt darf 160 Zeichen für SMS bzw. 80 Zeichen für CityRuf nicht überschreiten.

Im Body wird jede neue Zeile durch `<E _=“Text“/>` dargestellt.

4.1.4. Testen des Alarms (TiXML Reference Kapitel 2.4.9.1)

Der einfache Alarm ist nun vollständig parametriert. Nachdem das Projekt ins Alarm Modem geladen wurde, kann der Alarm über einen DoOn-Befehl an der seriellen Schnittstelle ausgelöst werden, z.B. `[<DoOn _=“Alarm_0“/>]`.

4.2. Erweiterte Alarmfunktionen

Eine einfache Alarmfunktion kann durch viele Funktionen erweitert werden.
Der Alarm kann z.B. über einen Port oder eine SPS-Variable ausgelöst werden, einen Port setzen, Alarmkaskadierungen enthalten oder sogar auf Betätigungen warten.

Die Kette der vom Tixi Alarm Modem abgearbeiteten Datenbanken wird dadurch, wie in Kapitel 1.2 beschrieben, erweitert:
EventStates – ProcessVars – EventHandler – MessageJobTemplates – AddressBook - MessageText

Dieses Kapitel wird den vorhandenen einfachen Alarm (Fax-Alarm) schrittweise erweitern.

4.2.1. Erweitern der Meldetexte

4.2.1.1. Signaturen (TiXML Reference Kapitel 3.6)

Ein großer Vorteil der Verwendung von XML ist die Möglichkeit über Referenzen Inhalte miteinander zu verknüpfen.
Als Beispiel (`basis-signature.cnf`) dient hier ein Text mit Standortdaten, der als Signatur an jede Nachricht angehängt werden soll. Dies spart Platz im Filesystem und Änderungen in der Signatur werden dann in allen Nachrichten wirksam. TILA2 kennt diese Funktion bereits.

Die Signatur:

```
<LocationText hidden="1">
      <Email>
            <E _="Signature of the Alarm Modem"/>
      </Email>
      <SMS _="Modem Signature"/>
</LocationText>
```

In den Text wird es per „Include“ eingefügt (bei Email, Fax, Express-E-Mail):

```
<Message_0 Name="Fax-Alarm" Type="Mail" UseSignature="1">
      <Subject _="This is the subject line of the fax."/>
      <Body>
            <E _="This is the body of the fax."/>
            <E _="Enter as many lines as you need."/>
            <E _=""/>
            <E _="This is the signature:"/>
            <E _="----------------------------------------"/>
            <Include _="/D/UserTemplates/LocationText/Email"/>
      </Body>
</Message_0>
```

Beim Empfänger erscheint dann dieser Text:

```
This is the subject line of the fax.

This is the body of the fax.
Enter as many lines as you need.

This is the signature:
---------------------------------------
Signature of the Alarm Modem
```

Bei SMS und CityRuf wird die Signatur über eine Referenz eingefügt:

```
<Message_1 Name="SMS-Alarm" Type="SMS" UseSignature="1">
     <Subject _="This is the message of the SMS Alarm. This is the
     signature: / &#xae;/D/UserTemplates/LocationText/SMS;"/>
</Message_1>
```

4.2.1.2. Text mit Variablen (TiXML Reference Kapitel 13)

Meldetexte können aber auch durch Variablen ergänzt werden, z.B. Systemvariablen wie Datum/Uhrzeit, Seriennummern oder Zustände von I/O-Ports oder SPS-Variablen. TILA2 bietet einen Teil dieser Variablen bereits an.
(`basis-fax-textvariables.cnf`)

Hinweis: Das Beispielprojekt erfordert ein Alarm Modem mit I/O-Erweiterung. Ohne diese Erweiterung muss die Zeile mit der Referenz auf den `Port I00` entfernt werden.

```
<Message_0 Name="Fax-Alarm" Type="Mail" UseSignature="0">
     <Subject _="This is the subject line. Port I00:
               &#xae;/Process/MB/IO/I/P0;"/>
     <Body>
          <E _="This is the message body."/>
          <E _="You can add variables to the text:"/>
          <E _="Date&amp;Time: &#xae;/TIMES/RFC822Date;"/>
          <E _="Firmware: &#xae;/OEM/Firmware/Version;"/>
          <E _=""/>
     </Body>
</Message_0>
```

4.2.2. Multiple Sendmail (TiXML Reference Kapitel 3.8)

Bisher war dem EventHandler ein einziger SendMail Befehl zugewiesen.
Es ist jedoch auch möglich mehrere SendMail Befehle in einem EventHandler zu verwenden, und so bei einem Alarm mehrere Empfänger und Nachrichtentypen zu bedienen (`basis-multiplesendmail.cnf`):

```
<Alarm_0 Name="Fax-Alarm">
     <SendMail _="MessageJobTemplates/Alarm_0"/>
     <SendMail _="MessageJobTemplates/Alarm_1"/>
</Alarm_0>
```

Jeder SendMail Befehl verweist hier auf ein eigenes MessageJobTemplate für Fax und E-Mail, diese wiederum jedoch auf einen gemeinsamen Nachrichtentext.

4.2.3. Port löst Alarm aus

4.2.3.1. Der Service-Button (TiXML Reference Kapitel 6.8)

Man kann den Service-Button an der Rückseite des Geräts zum Auslösen eines Alarms verwenden (`basis-fax-servicebutton.cnf`). Denkbar wäre auch die Verknüpfung mit einer Alarm-Quittierung (Kapitel 4.2.5.4).
Der Service-Button wird über einen System-Event-Handler eingebunden (Register `10-EventHandler`) :

```
<System>
      <OnButton>
            <SendMail _="MessageJobTemplates/Alarm_0"/>
      </OnButton>
</System>
```

TILA2 kann ebenfalls den Service-Button verwenden, spricht ihn aber in einem EventState (siehe 4.2.3.2) über die Systemvariable `/Process/MB/PollButton` an. Beide Wege sind möglich.

4.2.3.2. I/O-Ports (TiXML Reference Kapitel 6.3)

Um einen Alarm mit einem digitalen Eingang zu verknüpfen wird auf die EventStates Datenbank (Register `14-EventStates`) zurückgegriffen (`basis-messages-ports.cnf`):

Die EventStates verknüpfen einen EventHandler mit einer Prozess-Variable. Hier soll z.B. der Eingang P1 den Fax-Alarm auslösen:

```
<Alarm_0 Var="Product_0_1">
      <Event _="Alarm_0"/>
      <ProcessVar _="/Process/MB/IO/I/P1" flank="low"/>
      <Enabled _="TRUE"/>
</Alarm_0>
```

Mit dem Parameter "**Enabled**" kann ein EventState (de-)aktiviert werden.
Steht er auf TRUE, so nimmt das Tixi Alarm Modem den EventState in die Prozessabarbeitung auf, bei FALSE wird er ignoriert.

Der Parameter "**Event**" legt fest welcher Alarm beim Eintreten der Bedingung „ProcessVar" ausgelöst wird.

Der Parameter "**ProcessVar**" kann nun auf zwei verschiedenen Weisen konfiguriert werden, welche beide von TILA2 (wenn auch z.T. nur eingeschränkt) unterstützt werden:

1. **Direkt adressiert (schneller aber unflexibel)**

Bei der direkten Adressierung werden die Portadresse sowie der Wert der Bedingung im EventState eingegeben.

```
      <ProcessVar _="/Process/MB/IO/I/P1" flank="low"/>
```

In diesem Fall ist der Alarm mit dem Eingang P1 verknüpft. Der Parameter „low" legt fest, dass die Bedingung bei der Änderung des Port von 1 auf 0 eintritt. Alternativ gibt es die Parameter „high" (Wechsel auf 1) und „both" (jeder Wechsel)

2. Über ProcessVars Datenbank (langsamer, aber flexibler)

In diesem Fall weist der Parameter ProcessVar auf einen Eintrag in der ProcessVars Datenbank (siehe nächstes Kapitel):

```
<ProcessVar _="/Process/PV/Alarm_0_ProcVar"/>
```

4.2.3.3. ProcessVars (TiXML Reference Kapitel 6.2)

Die ProcessVars Datenbank (Register `13-ProcessVars`) legt die Bedingungen für EventStates fest (`basis-messages-processvars.cnf`).

Logische Operationen
Bei diesem Beispiel für den Fax-Alarm tritt die Bedingung immer dann ein, wenn der Port von 1 (offen) auf 0 (geschlossen) wechselt.

```
<Alarm_0_ProcVar sys="1">
      <Value>
            <LDN _="MB/IO/I/P0"/>
     </Value>
</Alarm_0_ProcVar>
```

Anstelle des LDN-Befehls können noch viele andere logische Operationen verwendet werden, z.B. LD, ST oder Operationen zum Verknüpfen mehrerer Ports:
Bei diesem Beispiel für den E-Mail-Alarm tritt die Bedingung immer dann ein, wenn beide Ports von 1 auf 0 wechseln.

```
<Alarm_1_ProcVar sys="1">
      <Value>
            <LDN _="MB/IO/I/P0"/>
            <ANDN _="MB/IO/I/P1"/>
     </Value>
</Alarm_1_ProcVar>
```

Vergleichsoperationen
Neben den logischen Operationen sind auch noch Vergleichsoperationen möglich. TILA2 verwendet diese Funktion bereits.
Hier für den Express-E-Mail-Alarm ein Beispiel mit einem Analogeingang des Modems. Man könnte Vergleiche aber auch mit SPS-Variablen verwenden.

```
<Alarm_2_ProcVar sys="1">
 <Value>
   <GE v1="/Process/MB/A/AI/P0" v2="5000"/>
   <LE v1="/Process/MB/A/AI/P0" v2="10000"/>
   <ANB _=""/>
 </Value>
</Alarm_2_ProcVar>
```

Die Bedingung ist aus zwei Vergleichen zusammengesetzt (Werte-Fenster) und tritt ein, wenn v1 größer gleich v2 (GE) und v1 kleiner gleich v2 ist (also im Bereich 5000-10000). Bei v2 kann der Wert auch über eine Referenz auf eine andere Variable gebildet werden.

Zeitvergleich
Eine weitere Bedingung für Prozessvariablen ist die Zeit, d.h. ein Alarm wird immer zu bestimmten Zeiten ausgelöst. Dabei können Uhrzeit und/oder Datum ein Kriterium sein. In diesem Beispiel erfolgt die SMS-Alarmierung täglich beim Wechsel in die angegebene Zeitspanne, also um 04:20.

```
<Alarm_3_ProcVar sys="1">
 <Value>
  <TIME _="04:20:00-06:30:00"/>
 </Value>
</Alarm_3_ProcVar>
```

Die Zeit-Bedingung kann auch mit logischen Bedingungen kombiniert werden, so lassen sich z.B. mehrere Zeiträume angeben:

```
<Alarm_4_ProcVar sys="1">
 <Value>
  <TIME _="04:20:00-06:30:00"/>
  <TIME _="08:00:00-10:00:00"/>
  <OR _=""/>
 </Value>
</Alarm_4_ProcVar>
```

Für komplexere zeitabhängige Alarmierungen ist die Scheduler-Datenbank zu verwenden (siehe nächstes Kapitel).

MSK / FIND_BIT
Einige Anwender von SPSen haben das Problem, dass zu überwachende Alarmbits nur als Bit in 16- oder 32Bit-Variablen zur Verfügung stehen. Diese müssen folglich zur Alarmierung aus einem Wort oder DWort gefiltert werden.
Ein ähnliches Problem ist die maximale Anzahl von Alarmen (EventStates), die im Alarm Modem derzeit auf 100 begrenzt ist. Möchte man wesentlich mehr Alarme auslösen, so kann man die Bits in der SPS zu Registern zusammenfassen, und im Modem wieder zerlegen. Somit sind theoretisch 100 x 32 (DWort) = 3200 Alarme möglich.

Die einfachste Lösung dafür ist die MSK-Anweisung. Bei dieser wird eine Bitmaske über das Wort oder DWort gelegt. Sobald ein oder mehrere Bits innerhalb der Maske gesetzt sind, ist die Prozessvariable 1 und kann somit einen Alarm auslösen.

```
<Alarm_0_ProcVar sys="1">
 <Value>
  <MSK v1="/Process/PV/PV1" v2="5"/>
 </Value>
</ Alarm_0_ProcVar >
```

In diesem Beispiel wird eine 16-Bit Prozessvariable PV1 mit einer Maske 5 gefiltert.

Die Prozessvariable besteht also aus 16 einzelnen Bits, die 0 oder 1 sein können:
Bit 16……..Bit 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Die Maske mit dem Wert 5 sieht in Dualschreibweise so aus:
Bit 16……..Bit 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1

d.h. Bit 1 (=1) und Bit 3 (=4) werden überwacht (1+4=5).

Sobald also Bit 1 oder Bit 3 oder beide gesetzt sind, ist die Prozessvariable 1.

Der Nachteil dieser Variante ist, dass nur festgestellt wird ob ein Bit innerhalb des Maske gesetzt ist, nicht jedoch welches. Dadurch lassen sich die Alarmtexte nicht auf das Bit beziehen.

Daher wurde die Anweisung FIND_BIT eingeführt. Diese funktioniert ähnlich wie MSK, liefert als Ergebnis jedoch keine 1, sondern die Position des gesetzten Bits.
(basis-messages-ports-FIND_BIT.cnf)

```
<Alarm_0_ProcVar>
 <Value>
  <LD _="/Process/PV/PV1"/>
  <FIND_BIT_ADDRESS _="1" range="1" mask="58759"/>
 </Value>
</Alarm_0_ProcVar>
```

Die Maske mit dem Wert 58759 sieht in Dualschreibweise so aus:
Bit 16……..Bit 1
1 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1

Die maskierten Bits werden vom System beginnend beim niedrigsten Bit fortlaufend nummeriert, im angegebenen Fall gibt es also die Bits 1 bis 9.
Sobald eins oder mehrere dieser Bits gesetzt ist, nimmt die Prozessvariable als Wert die Position (1-9) des ersten (bestimmt durch _="1") erkannten Bits an.
Lädt man dieselbe 16Bit-Variable noch mal in einer anderen Prozessvariable, und legt wieder die gleiche Maske drüber, setzt die Bitposition aber auf _="2", so erkennt diese das zweite gesetzte Bit innerhalb der Bitmaske. Somit lassen sich mehrere gleichzeitig gesetzte Bits innerhalb einer 16Bit-Variable bestimmen.

Beispiel:
Ist der der Wert der 16Bit Variable 128, so ist das Bit mit der Positionsnummer 4 gesetzt, die Prozessvariable nimmt also den Wert 4 an:

16 Bit Variable "128":	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Bitmaske "58759":	1	1	1	0	0	1	0	1	1	0	0	0	0	1	1	1
Bitpositionen:	9	8	7	-	-	6	–	5	4	-	-	-	-	3	2	1

Mit dieser Information kann man durch geschicktes Referenzieren den Alarmtext so dynamisch gestalten, dass je nach erkanntem Bit ein anderer Text gesendet wird.

Dazu legt man in den UserTemplates (Register 12 – MessageText) einen Satz Textbausteine für alle Bitpositionen an:

```
<TextTemplates>
 <BIT_0 _="No Bit detected !"/>
 <BIT_1 _="Error Bit 1 detected !"/>
 <BIT_2 _="Error Bit 2 detected !"/>
 <BIT_3 _="Error Bit 3 detected !"/>
 <BIT_4 _="Error Bit 4 detected !"/>
 <BIT_5 _="Error Bit 5 detected !"/>
 …
</TextTemplates>
```

Im Alarmtext werden diese Textbausteine abhängig vom Wert der Prozessvariablen per Referenz eingefügt, wobei ein Teil der Referenz durch eine Referenz auf die Prozessvariable ersetzt wird.

```
<Message_0 Name="SMS-Alarm" Type="SMS" UseSignature="0">
 <Subject
_="&#xae;/D/UserTemplates/TextTemplates/BIT_&#xae;/Process/PV/Alarm_0_ProcVar;;"/>
</Message_0>
```

Auslösen kann man den Alarm dann entweder über einen Vergleich der Prozessvariable größer Null (GT 0, s.o.),

```
<GT v1="/Process/PV/Alarm_0_ProcVar" v2="0"/>
```

oder über eine gleichzeitig mit dem Register veränderte Triggervariable der SPS (siehe Kapitel 4.2.3)

4.2.4. Scheduler (TiXML Reference Kapitel 7)

Der Scheduler (Register `16-Schedule`) wird für komplexe Zeitschaltfunktionen verwendet, z.B. Schichtpläne. Die Zeitkriterien reichen von Minuten, über Wochentage bis zu Monaten und Jahren. Kombinationen und Ausschlusszeiten (Ferienprogramm, Register `15-Schedule Conditions`) sind auch möglich.
(`basis-fax-scheduler.cnf`)

Dieser Scheduler verschickt z.B. Werktags um 18:00 den Fax-Alarm (denkbar als Tagesbericht):

```
<Schedule_0 _="Alarm_0">
 <Weekday _="Mo-Fr"/>
 <Time _="18:00"/>
</Schedule_0>
```

Der durch den Kalendereintrag auszulösende EventHandler (hier: Alarm_0), wird in der ersten Zeile des Eintrags als Parameter angegeben.

4.2.4.1. Schichtplan

Das Beispiel `basis-fax-Shift.cnf` demonstriert die Verwendung des Schedulers, um Alarmempfänger zeitabhängig zu wählen. Ab 09:00 UHR soll die Tagesschicht (Contact_0) den Fax-Alarm erhalten, ab 21:00 UHR erhält die Nachtschicht (Contact_1) die Alarme.

Zunächst wird der Empfänger der Nachtschicht im Adressbuch angelegt:

```
<Contact_1 Name="Shiftworker">
  <Fax _="+49-30-40608401"/>
  …
</Contact_1>
```

Des Weiteren wird eine Prozess-Variable benötigt, in welche der Scheduler den derzeit aktiven Empfänger schreiben kann (Register `13-ProcessVars`):

```
<Shiftworker sys="1"/>
```

Das Ändern dieser Prozess-Variable, und somit des Empfängers, wird über zwei EventHandler ausgeführt. Diese schreiben den Empfängernamen des Adressbuchkontaktes in die Variable:

```
<Switch_0>
 <Set _="/Process/PV/Shiftworker" value="Contact_0"/>
</Switch_0>

<Switch_1>
 <Set _="/Process/PV/Shiftworker" value="Contact_1"/>
</Switch_1>
```

Diese EventHandler werden nun zum Schichtwechsel über den Scheduler aufgerufen:

```
<Schedule_0 _="Switch_0">
 <Time _="09:00"/>
</Schedule_0>

<Schedule_1 _="Switch_1">
 <Time _="21:00"/>
</Schedule_1>
```

Damit der Alarm den Wert der Prozess-Variable als Empfänger verwendet, wird im dazugehörigen MessageJobTemplate der Pfad auf den Adreßbuchkontakt über die Prozess-Variable manipuliert:

```
<Alarm_0 _="TextFax">
 <Recipient _="/D/AddressBook/&#xae;/Process/PV/Shiftworker,Contact_0;"/>
 <Sender _="AddressBook/MySelf"/>
 <Subject _="&#xae;/D/UserTemplates/Message_3/Subject;"/>
 <Body _="/D/UserTemplates/Message_3/Body"/>
</Alarm_0>
```

Nach dem Auflösen der Referenz `&#xae;/Process/PV/Shiftworker;` findet das Alarm Modem je nach Tageszeit den Rezipienten:

`<Recipient _="/D/AddressBook/Contact_0"/>` oder
`<Recipient _="/D/AddressBook/Contact_1"/>`

Die im MessageJobTemplate durch ein Komma abgetrennte Angabe “Contact_0” dient als Start- bzw. Alternativwert. Sie wird immer dann verwendet, wenn die Prozess-Variable noch keinen Wert enthält, z.B. nach dem Neustart des Modems.

Die Variable kann nach dem Neustart aber auch manuell über einen Set-Befehl initialisiert werden:

```
[<Set _="/Process/PV/Shiftworker" value="Contact_0" ver="y"/>]
```

Es sei hier noch anzumerken, dass es eine einfachere Lösung für einen Schichtplan gibt, allerdings hätten Sie dann die freien Prozess-Variablen nicht kennengelernt:

Dazu benötigen Sie weder eine Prozessvariable, noch eine manipulierte Referenz im MJT. Sie lassen einfach den EventHandler den Adressbuch-Kontakt-Pfad per Set-Befehl direkt in das MessageJobTemplate schreiben:

```
<Switch_0>
 <Set _="/TEMPLATE/MessageJobTemplates/Alarm_0/Recipient"
      value="AddressBook/Contact_0"/>
</Switch_0>

<Switch_1>
 <Set _="/TEMPLATE/MessageJobTemplates/Alarm_0/Recipient"
      value="AddressBook/Contact_1"/>
</Switch_1>
```

Der Set-Befehl kann also auch zum Ändern von Datenbankeinträgen verwendet werden.

4.2.4.2. Scheduler Conditions (Ferienprogramm)

In vielen Fällen benötigt eine Kalenderfunktion, insbesondere im Fall des Schichtplans, Ausnahmeregelungen. Diese werden im Register `15-Schedule Conditions` definiert.

Als Beispiel dient der zuvor konfigurierte Schichtplan, in welchen die festen deutschen Feiertage eingefügt werden. An den Feiertagen soll ganztägig der Empfänger der Nachtschicht „Shift“ verwendet werden (`basis-fax-shift-condition.cnf`):

Um den gewünschten Effekt zu erreichen, wird an den Feiertagen der Wechsel auf die Tagesschicht weggelassen. Im Scheduler führt man dort eine Bedingung ein:

```
<Schedule_0 _="Switch_0">
 <Time _="09:00"/>
 <Condition _="Condition/Holidays"/>
</Schedule_0>
```

Diese Bedingung verweißt auf die Condition-Datenbank:

```
<Holidays>
 <Data not="1.1.,1.5.,3.10.,25.12.,26.12."/>
</Holidays>
```

Der Parameter „`not`“ legt fest, dass der Kalendereintrag an diesen Tagen nicht gültig ist.

Alternativ hätte diese Bedingung auch direkt im Schedulereintrag eingegeben werden können:

```
<Schedule_0 _="Switch_0">
 <Time _="09:00"/>
 <Data not="1.1.,1.5.,3.10.,25.12.,26.12."/>
</Schedule_0>
```

Der Vorteil der Verwendung der Condition-Datenbank ist aber, dass die Bedingung so für mehrere Kalenderfunktionen verwendet werden kann, und somit nur einmal eingegeben werden braucht, und zentral geändert werden kann.

4.2.5. erweiterte EventHandler Funktionen (TiXML Reference Kap. 3.8.1)

Im Tutorial wurde der EventHandler bisher nur zum Verschicken von Nachrichten per SendMail verwendet. Es sind allerdings noch viel komplexere Operationen möglich:

4.2.5.1. Alarmprioritäten

Jedem SendMail Befehl kann eine Priorität zugewiesen werden. Anhand der Priorität werden die zu versendenden Nachrichten in der Warteschlange sortiert, und zwar wird die Nachricht mit der höchsten Priorität (1=kleinste, 255=größte) zuerst verschickt.
Wenn also z.B. ein Feueralarm mit Priorität 10 ausgelöst wird, und zeitgleich noch einige Nachrichten mit niedrigerer Priorität in der Warteschlange stehen, so wird die Nachricht sofort nach dem frei werden der Leitung verschickt. (`basis-messages-priorities.cnf`)

Die Priorität wird im EventHandler wie folgt festgelegt:

```
<Alarm_0 Name="Fax-Alarm">
<SendMail _="MessageJobTemplates/Alarm_0">
  <Priority _="5"/>
 </SendMail>
</Alarm_0>
```

4.2.5.2. Alarm setzt Ausgang - Set

Per Set-Befehl können Ausgänge manuell gesetzt werden (TiXML Reference Kap. 6.5.3). Dieser Befehl kann jedoch auch in einen EventHandler eingebaut werden.
(`basis-fax-setport.cnf`)
In diesem Beispiel wird neben dem Versenden der Nachricht auch noch der Ausgang P2 auf 1 gesetzt (denkbar als Alarmkontakt):

```
<Alarm_0 Name="Fax-Alarm">
 <SendMail _="MessageJobTemplates/Alarm_0"/>
 <Set _="/Process/MB/IO/Q/P2" value="1"/>
</Alarm_0>
```

4.2.5.3. Alarmkaskade „OnError“

Das Tixi Alarm Modem kann mit Wahlwiederholungen arbeiten. Diese helfen jedoch in dem Fall nicht weiter, in dem ein Empfänger einfach nicht erreichbar ist.
Für diesen Fall sind Alarmkaskaden gedacht, welche bereits von TILA2 in einfacher Form angeboten werden. Wenn eine Nachricht nicht abgesetzt werden konnte wird sie einfach an einen anderen Empfänger geschickt (`basis-cascade.cnf`):

```
<Alarm_0 Name="Fax-Alarm">
<SendMail _="MessageJobTemplates/Alarm_0">
  <OnError _="Alarm_1"/>
  <MaxRepeat _="1"/>
  <Interval _="180s"/>
 </SendMail>
</Alarm_0>
```

In diesem Beispiel wird eine Nachricht zweimal versucht zuzustellen (MaxRepeat=1 bedeutet eine Wahlwiederholung mit Interval=180s: 180s Abstand). Wenn beide Versuche fehlschlagen wird das Event „Alarm_1“ ausgelöst.
Es wäre nun denkbar das Event „Alarm_1“ auch wieder mit einer Kaskade auszustatten. Es könnte sogar zurück auf das Event „Alarm_0“ verweisen und so eine Schleife erstellen, die erst nach dem ordnungsgemäßen Absetzen der Nachricht abbricht.

4.2.5.4. Bestätigung über Port – „OnOK“

Neben dem Befehl „OnError“ gibt es auch noch den Befehl „OnOK“ der z.B. zum Schalten von Ausgängen in Verbindung mit einer erfolgreich abgesetzten Störmeldung verwendet werden kann (`basis-cascade.cnf`).
In diesem Beispiel wird im Alarmfall ein Ausgang gesetzt und nach dem erfolgreichen Absetzten der Nachricht der Port über das Event “Switch_0“ zurückgesetzt.

```
<Alarm_2 Name="Express-E-Mail-Alarm">
 <Set _="/Process/C0/Q/P2" value="1"/>
 <SendMail _="MessageJobTemplates/Alarm_2">
  <OnOK _="Switch_0"/>
  <MaxRepeat _="1"/>
  <Interval _="180s"/>
 </SendMail>
</Alarm_2>
<Switch_0 Name="Switch_0">
<Set _="/Process/MB/IO/Q/P2" value="0"/>
</Switch_0>
```

Es können auch „OnOK“ und „OnError“ Befehle gemeinsam in einem SendMail-Befehl verwendet werden.

4.2.5.5. Quittieren von Nachrichten

Eine Alternative zur Alarmkaskade ist das Verschicken von Nachrichten mit Quittierungsanforderung. Dadurch ist gewährleistet, daß die Alarmnachricht den Empfänger tatsächlich erreicht hat.
Bestätigungen sind per SMS, Express-E-Mails, Internet E-Mail (eingeschränkt) oder Variablen möglich.
TILA2 bietet derzeit nur die Quittierung über SMS.
(`basis-confirmation.cnf`)

Der EventHandler sieht wie folgt aus:

```
<Alarm_0 Name="SMS-Alarm">
      <SendMail _="MessageJobTemplates/Alarm_0">
            <OnError _="Alarm_0"/>
            <OnTimeout _="Alarm_0"/>
            <ConfirmID _="0"/>
            <Timeout _="2m"/>
      </SendMail>
</Alarm_0>
```

Der dazugehörige Nachrichtentext:

```
<Message_0 Name="SMS-Alarm" Type="SMS" UseSignature="0">
      <Subject _="SMS-Alarm with acknowledge request. Code:
      &#xae;~/_Fingerprint;"/>
</Message_0>
```

Zudem ist ein spezieller System-EventHandler nötig:

```
<System>
   <Confirmation>
      <Confirm _="&#xae;~/_ConfirmID;"/>
   </Confirmation>
</System>
```

Die Quittierungsfunktion arbeitet mit einer BestätigungsID (ConfirmID, im Beispiel “0“) und einem eindeutigen, automatisch generierten Fingerprint. Jedem Alarm muss im EventHandler eine eigene Confirm-ID zugeordnet werden.
Der Fingerprint wird über eine Referenz in den Betreff der Nachricht integriert, und muss als Quittierung lediglich vom Empfänger zurückgeschickt werden.

Der Befehl „OnTimeout“ legt dabei fest, was nach Ablauf der Timeout-Zeit, in der die Quittierung erfolgen muss, geschieht. Im Beispiel wird der gleiche Event noch mal ausgelöst, so dass es solange wiederholt wird, bis eine Quittierung eingetroffen ist.
Denkbar wäre auch das Auslösen eines anderen Events mit alternativem Empfänger nach Ablauf des Timeouts.

4.2.5.6. Übergabe von Parametern

Wenn ein Alarm per „DoOn“-Befehl ausgelöst wird können ihm auch Parameter übergeben werden, die z.B. in Meldetexte eingebaut werden können. (`basis-fax-parameter.cnf`)

Als Beispiel dient der Fax-Alarm. In den Nachrichtentext soll eine Zimmernummer dynamisch eingefügt werden.

```
<Message_3 Name="Fax-Alarm" Type="Mail" UseSignature="0">
     <Subject _="This is the subject line of the Fax."/>
     <Body>
          <E _="This is the message body of the Fax."/>
          <E _="Enter as many lines as you need."/>
          <E _=""/>
          <E _="Value parameter room: &#xae;~/Room;"/>
     </Body>
</Message_3>
```

Hier wird im Text der beim DoOn übergebene Parameter “Room“ über eine Referenz ®~/Room; eingefügt.

Der **DoOn-Befehl** muss dazu wie folgt aussehen:

```
[<DoOn _="Alarm_0" ver="y">
     <Room _="23.3"/>
</DoOn>]
```

Das generierte Fax sieht dann so aus:

```
This is the subject line of the Fax.

This is the message body of the Fax.
Enter as many lines as you need.

Value parameter room: 23.3
```

Die Anzahl der Parameter ist beliebig. Es ist auch denkbar, die übergebenen Parameter als Wert für Variablen zu verwenden.
So könnte z.B. ein EventHandler mit „**Set-Befehl**“ einen Port auf den übergebenen Wert setzen:

DoOn-Befehl:

```
[<DoOn _="Switch_0" ver="y">
     <Parameter _="1"/>
</DoOn>]
```

EventHandler:

```
<Switch_0>
     <Set _="/Process/MB/IO/Q/P2" value="&#xae;~/Parameter;"/>
</Switch_0>
```

4.2.5.7. Logikoperationen und Berechnungen

Die Logikoperationen, die Sie bereits im Kapitel 4.2.3.3 zur Berechnung von Prozessvariablen kennengelernt haben, können auch direkt ein einem EventHandler verwendet werden. Dies ermöglicht zum Beispiel das einfache Umkopieren von Werten (`basis-process.cnf`):

Im EventHandler (Register 10 – EventHandler) wird dazu die Anweisung „Process“ eingefügt. Diese kann dann die selben Logik- und Vergleichsoperationen enthalten, die auch bei Prozessvariablen erlaubt sind:

```
<Alarm_0>
 <Process>
  <LD _="/Process/MB/IO/I/P0"/>
  <ST _="/Process/MB/IO/Q/P2"/>
 </Process>
</Alarm_0>
```

Im gezeigten Beispiel wird beim Aufruf des Events der Wert des Eingangs P0 in den Ausgang P2 kopiert.

Möchte man zu bestimmten Zeitpunkten (also Eventgesteuert) Berechnungen ausführen, so bietet sich die Verwendung von mathematischen Operationen ADD,SUB,MUL und DIV im EventHandler an (`basis-process-calculator.cnf`):

```
<Alarm_1>
 <Process>
  <LD _="/Process/PV/Counter"/>
  <ADD _="1"/>
  <ST _="/Process/PV/Counter"/>
 </Process>
</Alarm_1>
```

Vorliegendes Beispiel erzeugt einen Zähler, der bei jedem Aufruf eine Prozessvariable um 1 erhöht. Kombiniert man diesen Event z.B. mit dem Scheduler, kann auf diese Weise ein Betriebsstundenzähler realisiert werden.

4.2.5.8. Loggen von Daten (TiXML Reference Handbuch Kapitel 4)

Neben den bereits bekannten Logfiles können auch eigene Logfiles erzeugt und mit Daten beschrieben werden. Diese können zum einen als XML-Logfiles, zum anderen als platzsparende Binärlogfiles angelegt werden. Letztere Variante ist in jedem Fall vorzuziehen, weshalb das XML-Logging hier nicht weiter erwähnt wird.

Binär-Logging (`basis-portbinlog.cnf`):

Für binäres Datenloggen wird als erstes ein **binäres Logfile** der gewünschten Größe (Ringspeicher) erzeugt. Das Logfile verweist dort auf ein Record, in welchem die Datenstruktur beschrieben ist (Register `7/8-LogDefinition`):

```
<LogDefinition>
  <LogFiles>
     <JobReport size="10240"/>
       <Event size="10240"/>
       <Login size="10240"/>
       <IncomingMessage size="10240"/>
       <FailedIncomingCall size="10240"/>
          <Logfile_0 size="10240" contenttype="binary"
          record="Record_0"/>
  </LogFiles>
  <Records>
    <Record_0>
      <Value1 _="word" value="&#xae;/Process/MB/A/AI/P0;"/>
    </Record_0>
  </Records>
</LogDefinition>
```

Hier wird der Typ der Daten (“byte“, “word“, “dword“, “int“, “string“ usw.), ggf. die Größe des Datensatzes (z.B. size=“7“ bei string) sowie der zu loggende Wert (via Referenz auf Analogeingang P0) angegeben.

Im **EventHandler** muss lediglich der Befehl “BinLog“ mit dem zu verwendenden Logfile aufgerufen werden:

```
<Logging_0>
     <BinLog _="Logfile_0"/>
</Logging_0>
```

Zeitgesteuert Daten loggen:

Damit die Logging-Funktion einen Sinn macht, sollte diese über den Scheduler (Kapitel 4.2.4) zyklisch ausgeführt werden, z.B. alle 5 Minuten:

```
<Schedule_0 _="Logging_0">
     <Minute _="0,5,10,15,20,25,30,35,40,45,50,55"/>
</Schedule_0>
```

Da alle Logfiles als Ringpuffer verwendet werden, kann das Filesystem nicht volllaufen.

Senden von Daten als Attachment:

In vielen Fällen kann der Empfänger mit den geloggten Daten in XML-Format nicht viel anfangen. Zur Weiterverarbeitung der Daten bietet sich das CSV-Format an, welches z.B. in Excel gelesen werden kann. Via „IncludeLogTXT“ Befehl können die Daten vor dem Einfügen formatiert werden, z.B. die IDs, Datum und Uhrzeit entfernt oder die Zeilen durch Start- und End- und Trennzeichen ergänzt werden. Zudem kann das Logfile in ein Attachment mit beliebigen Namen verpackt werden.
(`basis-portbinlog-email-attachment.cnf`).

Zunächst wird definiert, wie das Attachment aussieht (z.B. CSV), und welche Daten dort eingefügt werden sollen (Register 12-Message Text):

```
<Attachments_0 hidden="1">
 <Attachment filename="Logfile_0.csv">
   <IncludeLogTXT _="Logfile_0" range="previous 24 hours"
     type="CSV"/>
 </Attachment>
</Attachments_0>
```

Bei dem Parameter `filename="Logfile_0.csv"` kann anstelle von `Logfile_0.csv` jeder beliebige Dateiname angegeben werden. Denkbar wäre auch, den Dateinamen aus dem aktuellen Datum zu generieren: `filename="&#xae;/TIMES/Date;.csv"`

Beim IncludeLogTXT Befehl wird der Name des Logfiles sowie der Bereich “range“ der zu sendenden Daten festgelegt. Im Beispiel werden die Daten der letzten 24 Stunden verschickt. Die möglichen Bereiche sind im TiXML Reference Handbuch im Kapitel 2.4.9.2 beschrieben.

Der Attachment-Block kann auch mehrere Dateien (Attachment-Gruppen) beinhalten.

Die umfangreichen Formatierungsmöglichkeiten (z.B. HTML, XML usw.) sind dem TiXML-Reference Handbuch Kapitel 4.9 zu entnehmen.

Anschließend wird dem MessageJobTemplate dieser Attachment-Block zugewiesen:

```
<Alarm_1 _="SMTP">
 <Recipient _="AddressBook/Contact_0"/>
 <Sender _="AddressBook/MySelf"/>
 <Subject _="&#xae;/D/UserTemplates/Message_4/Subject;"/>
 <Body _="/D/UserTemplates/Message_4/Body"/>
 <Attachments _="/D/UserTemplates/Attachments_0"/>
</Alarm_1>
```

Regelmäßiger Logfileversand per Scheduler:

```
<Schedule_1 _="Alarm_1">
 <Time _="00:01"/>
</Scheule_1>
```

4.2.5.9. IF-Bedingung (TiXML-Reference Handbuch Kapitel 3.8.2)

Möchte man einen EventHandler-Befehl mit einer Bedingung verknüpfen, um z.B. SPS-Daten nur dann zu loggen, wenn die SPS auch wirklich angeschlossen ist (DeviceState=1), so lässt sich dies am einfachsten über die IF-Anweisung realisieren (`basis-portbinlog-IF-condition.cnf`):

Um den oder die EventHandler Befehle wird das Tag `"If"` gelegt, welches im Starttag einen Pfad zu einer Bitvariable aufweist (Register 10 – EventHandler):

```
<Logging_0>
 <If _="/Process/MB/IO/I/P0">
  <BinLog _="Logfile_0"/>
 </If>
</Logging_0>
```

Die umschlossenen Befehle werden nur dann ausgeführt, wenn die Bitvariable auf `"1"` steht. Möchte man die Befehle bei „0“ ausführen, muss die Bitvariable in einer Prozessvariable negiert werden (siehe Kapitel 4.2.3.3).

Es ist zu beachten, das der Pfad in der IF-Bedingung ohne Referenz (®) eingegeben wird.

4.2.5.10. Internetzeitsynchonisierung / Sommer-Winter-Zeitumstellung (TiXML-Reference Kap. 3.13)

Die RTC des Tixi Alarm Modems kann über einen Zeitserver im Internet z.B. mit der Zeit der Atomuhr abgeglichen werden. Diese Funktion kann daher auch zur automatischen Sommer-Winter-Zeitumstellung verwendet werden.

Es gibt zwei verschiedene Zeitserverprotokolle:
TIME liefert die UTC Zeit in einem standardisierten String
DAYTIME liefert die lokale Zeit in einem nichtstandardisierten String

Beim TIME-Protokoll muss die Umrechnung der Zeitzone (z.B. +0100 für Deutschland) selber durchgeführt werden. Die Sommer-Winterzeit ist ebenfalls noch nicht enthalten.

Beim DAYTIME-Protokoll liefert der Server die lokal gültige Uhrzeit inkl. Zeitzonenumrechnung und Sommer-Winterzeit. Dem Alarm Modem muss das Format des Strings vorgegeben werden. Da die Zeit bereits vollständig umgerechnet vorliegt, müssen die TimeZone und der TimeDiff-Parameter auf +0000 stehen.

Der Zeit-String eines Internet-TimeServers kann über eine Abfrage des Port 13 erfolgen. Wenn man mit dem Internet verbunden ist, kann über die Eingabeaufforderung der String wie folgt abgefragt werden:
`telnet Server-URL 13`, z.B.
`telnet ptbtime1.ptb.de 13`

Als Antwort erhält man z.B.:
`10 JUN 2004 10:24:16 METDST`

Das Beispiel `basis-timeserver.cnf` zeigt die Verwendung des deutschen Atomuhr-DAYTIME-Servers zur Sommer-Winterzeitumstellung mit Hilfe des Schedulers (siehe Kapitel 4.2.4).

Timeserver-Einstellungen
Im Register `4-ISP` steht die URL des Timeservers, das Protokoll, die Zeitdifferenz sowie ggf. das Zeit-Format:

```
<TimeServer>
  <ServerName _="ptbtime1.ptb.de"/>
  <Protocol _="DAYTIME"/>
  <TimeDiff _="+0000"/>
  <TimeFormat _="d E y h:n:s "/>
</TimeServer>
```

TimeZone
Im Register `2-UserData` ist die TimeZone auf +0000 gesetzt:
`<TimeZone _="+0000"/>`

EventHandler
Der EventHandler `SyncTime` auf Register `10-EventHandler` ruft den Befehl `INetTime` zur Internetzeitsynchronisierung auf:

```
<SyncTime>
  <INetTime/>
</SyncTime>
```

Scheduler
Die Zeitsynchronisierung muss nun noch zu sinnvollen Zeitpunkten automatisch vom Alarm Modem ausgeführt werden, z.B. am ersten jeden Monats:

```
<Schedule_0 _="SyncTime">
  <Month _="1-12"/>
</Schedule_0>
```

Um die Sommer- und Winterzeitumstellung zu realisieren, muss das Modem sich am entsprechenden Tag kurz nach der Zeitumstellung mit dem Server synchronisieren.
Europaweit erfolgt dies einheitlich am letzten Sonntag des März und Oktober.
Das ergibt folgende Scheduler-Konfiguration:

```
<Summertime _="SyncTime">
  <Month _="3"/>                    im März
  <Day _="25-31"/>                  letzten 7 Tage
  <Weekday _="Su"/>                 am Sonntag
  <Time _="02:00"/>                 um 2 UHR
</Summertime>

<Wintertime _="SyncTime">
  <Month _="10"/>                   im Oktober
  <Day _="25-31"/>                  letzten 7 Tage
  <Weekday _="Su"/>                 am Sonntag
  <Time _="03:00"/>                 um 3 UHR
</Wintertime>
```

4.2.6. Sequencer (TiXML-Reference Handbuch Kapitel 8)

Möchte man zu bestimmten Zeiten bestimmte Werte setzen, z.B. tageszeitabhängige Grenzwertvergleiche (Lastprofile), so lässt sich das komfortabel mit dem Sequencer realisieren. Bei diesem werden dem Modem die Zeitpunkte und Werte in einer Art Tabelle übergeben (`basis-sequencer.cnf`).

Die Tabelle kann im XML-Format oder im CSV-Format (mit TiXML-Frame) per TiXML-Befehl `SetSequence` übergeben werden. Letztere Variante ermöglicht den einfachen Import aus Excel.

Sequenz im XML-Format:

```
[<SetSequence _="Sequence_0" priority="0" ver="v">
<T date="*.*.*" time="*:00" P1="1" P2="0" P3="1"/>
<T date="*.*.*" time="*:15" P1="0" P2="1" P3="0"/>
<T date="*.*.*" time="*:30" P1="1" P2="0" P3="1"/>
<T date="*.*.*" time="*:45" P1="0" P2="1" P3="0"/>
</SetSequence>]
```

Sequenz im CSV-Format mit TiXML-Frame:

```
[<SetSequence _="Sequence_0" mode="text">
<![CDATA[
!priority=1
!mask="d;t;1;2;3"
*.*.*;*:00;1;0;1
*.*.*;*:15;0;1;0
*.*.*;*:30;1;0;1
*.*.*;*:45;0;1;0
]]>
</SetSequence>]
```

Jede Sequenzanweisung enthält ein Datum im Format DD.MM.YYYY, eine Uhrzeit im Format HH:MM sowie eine Liste von bis zu 6 Werten. Teile des Datums und der Uhrzeit können durch Asterisk (*) ersetzt werden, sodass das Datum “*.*.*“ für “jeden Tag“ und die Uhrzeit “*:15“ für “15min nach jeder vollen Stunde“ steht.

Definiert werden die dazugehörigen Sequenzen im „Register 21 – Sequencer“:

```
<Sequencer>
 <Sequence_0 event="Switch_0" logfile="Sequence_0_Log"/>
</Sequencer>
```

Der Sequenz wird ein EventHandler zugewiesen (`event`) sowie ein Logfile (`logfile`).

In das Logfile werden übergebene Sequenzen zur Kontrolle abgespeichert.

Der EventHandler wird zu den angegebenen Sequenzzeitpunkten aufgerufen, und ihm die bis zu 6 Werte als P1 bis P6 übergeben.

```
<Switch_0>
 <Set _="/Process/MB/IO/Q/P0" value="&#xae;~/P1;"/>
 <Set _="/Process/MB/IO/Q/P1" value="&#xae;~/P2;"/>
 <Set _="/Process/MB/IO/Q/P2" value="&#xae;~/P3;"/>
</Switch_0>
```

4.3. Fernwirken durch eingehende Nachrichten (H. Kap. 9)

Das Gerät ist auch in der Lage E-Mails, Express-E-Mails und SMS zu empfangen und auszuwerten. So können z.B. Events ausgelöst werden um I/O-Ports oder SPS-Variablen zu ändern. TILA2 beherrscht derzeit den Empfang von SMS und Express-E-Mails.
Das Beispiel bezieht sich auf den Empfang von SMS (`basis-remoteswitch.cnf`):

4.3.1. Zugriffschutz

Zunächst wird ein Zugriffsschutz auf dem Register `5/6-AccRights` konfiguriert, damit nur autorisierte Personen Fernwirken können.

Einfacher Zugriffschutz:
Als Passwort wird z.B. das Wort „TIXI“ gewählt:

```
<AccRights>
      <Groups>
            <MessageGroup>
                  <Message AccLevel="1"/>
            </MessageGroup>
      </Groups>
      <User _="Plain">
            <Def_Message Plain="TIXI" Group="MessageGroup"/>
      </User>
</AccRights>
```

Zugriffschutz mit Nummernüberprüfung (SMS, Express-E-Mail):
Wird anstelle von “`Def_Message`“ das Kürzel “`OA_`“ gefolgt von einer Nummer eingetragen, z.B.:

```
      <User _="Plain">
            <OA_491721234567 Plain="TIXI" Group="MessageGroup"/>
      </User>
```

so wird neben dem Passwort auch die übermittelte Rufnummer überprüft, d.h. das Fernwirken ist nur von der angegebenen Rufnummer aus möglich.

Im GSM-Netz wird die Rufnummer mit „+Landeskennzahl“ übermittelt (z.B. +491721234567). Das „+“ wird beim OA nicht mit angegeben.

Zugriffschutz mit Absenderüberprüfung (Internet E-Mail):
Bei E-Mail-Adressen wird der Alpha-Parameter (E-Mail Alias) ausgewertet:
Aus der Absenderkennung “`Tixi Support`“ ergibt sich folgender Eintrag (Leerzeichen durch Unterstriche ersetzen!):

```
      <User _="Plain">
            <Tixi_Support Plain="TIXI" Group="MessageGroup"/>
      </User>
```

4.3.2. Einfaches Fernwirken:

Mit dem angegebenen Passwort kann jedes im Gerät konfigurierte Event (EventHandler) ausgelöst werden. Dazu muss der Betreff (E-Mail, Express-E-Mail) bzw. der SMS-Text der Nachricht das Passwort und den Eventnamen in folgender Syntax enthalten:

```
Passwort Eventname
```
z.B.

```
TIXI SETPORT
```

Bei diesem Beispiel lautet das Passwort „TIXI“ und der EventHandler „SETPORT“.
Der dazu passende **EventHandler** könnte so aussehen:

```
<SETPORT Name="SETPORT">
      <Set _="/Process/MB/IO/Q/P0" value="1"/>
</SETPORT>
```

Bei der eingehenden SMS wird also der Ausgang P0 auf 1 gesetzt.

4.3.3. Fernwirken mit Parameterübergabe

Um einem durch Fernwirken ausgelösten Event Parameter zu übergeben, muss der Betreff (E-Mail, Express-E-Mail) bzw. der SMS-Text der Nachricht das Passwort, den Eventnamen und die Parameter in folgender Syntax enthalten:

```
Passwort Eventname Parameter1 Parameter2 ... Parameter6
```
z.B.

```
TIXI SETVALUE 1
```

Bei diesem Beispiel lautet das Passwort „TIXI“ und der EventHandler „SETVALUE“. Dem Event wird der Parameter „1“ zur Weiterverarbeitung übergeben.

Der dazu passende **EventHandler** könnte so aussehen:

```
<SETVALUE Name="SETVALUE">
      <Set _="/Process/MB/IO/Q/P2" value="&#xae;~/P1;"/>
</SETVALUE>
```

Der Parameter wird über die Referenz `&#xae;~/P1;` (bei mehreren Parametern bis P10 hochzählen) dem Set-Befehl als neuer Wert für den Ausgang P2 übergeben, der Port wird also auf 1 gesetzt.
Mit der SMS `TIXI SETVALUE 0` könnte der Port wieder auf 0 werden.

4.3.4. Überwachung der Fernwirkfunktionen

In vielen Fällen ist es ratsam das Fernwirken durch Rückantworten zu erweitern.
Den Event erweitert man dafür um einen Sendmail Befehl, der nach dem Set-Befehl ausgeführt wird: (`basis-remoteswitch-answer.cnf`)
Das Beispiel ist für ein Tixi Alarm Modem GSM.

```
<SETPORT Name="SETPORT">
      <Set _="/Process/MB/IO/Q/P0" value="1"/>
      <SendMail _="MessageJobTemplates/Switch_0"/>
</SETPORT>
```

4.3.4.1. Statische Rückantwort

Als **Bestätigung** für den Vorgang wird eine Nachricht an einen Empfänger aus dem Adressbuch erstellt. Das dazu passende MessageJobTemplate sieht wie folgt aus:

```
<Switch_0 _="GSMSMS">
     <Recipient _="AddressBook/Contact_0"/>
     <Sender _="AddressBook/MySelf"/>
     <Subject _="&#xae;/D/UserTemplates/Message_0/Subject;"/>
</Switch_0>
```

In den Nachrichtentext kann zur weiteren Sicherheit eine Referenz auf den ausgeführten Event-Namen eingebaut werden:

```
<Message_0 Name="SMS Answer" Type="SMS" UseSignature="0">
     <Subject _="The Event &#xae;~/Event; was successfully
     processed."/>
</Message_0>
```

4.3.4.2. Dynamische Rückantwort

Um das Fernwirken dynamischer zu gestalten, kann die Rückmeldung auch an den Absender der Fernwirknachricht erfolgen (Beispielevent: SETVALUE).
Das MessageJobTemplate verweist in diesem Fall nicht auf einen Empfänger im Adressbuch, sondern erhält die Empfängeradresse bzw. –nummer aus der Absenderkennung der eingegangenen Nachricht:

Das dazu passende MessageJobTemplate verweist auf den System-Adreßbucheintrag “OA“, welcher über eine Referenz die empfangene Absenderkennung als Empfänger für die Rückmeldung verwendet:

```
<Switch_1 _="GSMSMS">
     <Recipient _="AddressBook/OA"/>
     <Sender _="AddressBook/MySelf"/>
     <Subject _="&#xae;/D/UserTemplates/Message_0/Subject;"/>
</Switch_1>
```

Adreßbuch:

```
<OA hidden="1">
     <SMS_No _="&#xae;~/OA"/>
     <Email _="&#xae;~/OA"/>
     <Express-Email _="&#xae;~/Alpha"/>
     <SMS_Provider _="AnnyWay"/>
</OA>
```

Es ist zu beachten, dass die Referenz auf den Empfänger je nach Nachrichtentyp variiert.
Bei SMS: `&#xae;~/OA;`
Bei E-Mail oder Express-E-Mail: `&#xae;~/Alpha;`

Wichtiger Hinweis für SMS-Empfang bei Alarm Modems am Festnetz (V.90/ISDN):

Der SMS-Empfang im Festnetz erfordert einen Telekom-Telefonanschluß mit Rufnummernanzeige (CLIP/CLIR).

Die dynamische SMS-Rückantwort über den Parameter „OA“ funktioniert nur bei GSM Alarm Modems überall, da diese über das ServiceCenter der SIM-Karte an alle Handy-Netze SMS schicken können.
Festnetz-Modems müssen je nach Empfänger-Netz ein anderes Gateway anwählen, und dieses kann anhand der Absender-Rufnummer nicht mehr eindeutig zugeordnet werden (Rufnummernmitnahme bei Providerwechsel).

Das für den Versand der Rückmeldung zuständige Gateway muss im Eintrag `<SMS_Provider _=“”/>` des Adreßbucheintrags “OA“angepasst werden. Für Deutschland wird „AnnyWay“ (alle Netze) empfohlen.

4.3.4.3. Rückantwort mit Wertüberprüfung

Zur weiteren Sicherheit kann in die Rückantwort neben dem ausgeführten Event-Namen auch der aktuelle Wert der veränderten Variable übermittelt werden. Dadurch kann überprüft werden ob der Schaltbefehl auch tatsächlich wirksam war.
Um den aktuellen Wert in der Nachricht zu erhalten, muss im EventHandler zwischen dem Set-Befehl und dem SendMail eine kleine Verzögerung eingefügt werden, da durch die schnelle Abarbeitung im System sonst ggf. noch der Wert vor wirksam werden des Set-Befehls ermittelt wird.

Der EventHandler sieht dann wie folgt aus:

```
<SETVALUE2 Name="SETVALUE2">
     <Set _="/Process/MB/IO/Q/P2" value="&#xae;~/P1;"/>
     <Delay _="5s"/>
     <SendMail _="MessageJobTemplates/Switch_2"/>
</SETVALUE2>
```

Der Nachrichtentext der Rückantwort mit Angabe des aktuellen Zustands von P2:

```
<Message_1 Name="SMS Answer Value" Type="SMS" UseSignature="0">
     <Subject _="The Event &#xae;~/Event; was successfully
     processed. Value Q2: &#xae;/Process/MB/IO/Q/P2;"/>
</Message_1>
```

4.3.5. Verhalten im Fehlerfall

Neben dem erfolgreichen Abarbeiten einer eingehenden Nachricht sind auch noch zwei Fehlerfälle denkbar:

- ungültiges Passwort
- ungültiges Event

Beide Fälle werden durch so genannte System-Events in der EventHandler Datenbank verarbeitet (`basis-remoteswitch-errors.cnf`):
Es ist zu beachten, dass diese Events in einer eigenen Untergruppe „`System`“ innerhalb der EventHandler-Gruppe stehen.

4.3.5.1. ungültiges Passwort

Falls eine empfangene SMS ein ungültiges Passwort enthält, wird automatisch der Event „`SMSInvalidPassword`“ bzw. „`TixiInvalidPassword`“ bei Express-E-Mails oder „`POPInvalidPassword`“ bei Internet E-Mails ausgeführt.

Diese Systemevents können beliebige EventHandler Befehle enthalten, z.B. Set- oder SendMail-Befehle.

Ein sinnvolles Beispiel wäre das Loggen des Absender (Parameter „OA“) und des empfangenen Textes (Parameter „Text“) zusammen mit der Meldung „SMS with invalid password received“:

```
<SMSInvalidPassword>
  <Log _="FailedIncomingCall">
    <Annotation _="SMS with invalid password received"/>
    <OA _="&#xae;~/OA;"/>
    <Text _="&#xae;~/Text;"/>
  </Log>
</SMSInvalidPassword>
```

Bei E-Mails und Express-E-Mails sollte zusätzlich der Parameter „Alpha“ (Absenderadresse) geloggt werden.

4.3.5.2. ungültiges Event

Falls eine empfangene SMS einen ungültigen Eventnamen enthält, wird automatisch der Event „`SMSInvalidEvent`“ bzw. „`TixiInvalidEvent`“ bei Express-E-Mails oder „`POPInvalidEvent`“ bei Internet E-Mails ausgeführt.

Diese Systemevents können beliebige EventHandler Befehle enthalten, z.B. Set- oder SendMail-Befehle.

Ein sinnvolles Beispiel wäre eine Rückmeldung an den Absender, sodaß er über den Fehler unterrichtet wird, sowie das Loggen des Absender (Parameter „OA“) und des empfangenen Textes (Parameter „Text“) zusammen mit der Meldung „SMS with invalid event received“:

```
<System>
   <SMSInvalidEvent>
   <Log _="FailedIncomingCall">
      <Annotation _="SMS with invalid event received"/>
      <Sender _="&#xae;~/OA;"/>
      <Text _="&#xae;~/Text;"/>
   </Log>
   <SendMail _="MessageJobTemplates/SMSAnswerOnError"/>
   </SMSInvalidEvent>
</System>
```

Bei E-Mails und Express-E-Mails sollte zusätzlich der Parameter „Alpha“ (Absenderadresse) geloggt werden.

Die Fehlermeldung wird am sinnvollsten, wie in Kapitel 4.3.4.2 beschrieben, an den Absender der Fernwirknachricht zurückgeschickt, und sollte einen Hinweis auf den unbekannten Event-Namen enthalten:

```
<Message_0 Name="SMS-Alarm" Type="SMS" UseSignature="0">
 <Subject _="Error: The Event &#xae;~/Event; is unknown."/>
</Message_0>
```

4.3.5.3. Informationen bei OK-Fall loggen

Wenn der ausgeführte EventHandler mit dem Log-Befehl erweitert wird, können auch im OK-Fall nützliche Daten des Fernwirkprozesses geloggt werden, z.B. die Absenderrufnummer und der Fernwirktext:

```
<SETPORT>
  <Log _="IncomingMessage">
    <OA _="&#xae;~/OA;"/>
    <Text _="&#xae;~/Text;"/>
  </Log>
  <Set _="/Process/MB/IO/Q/P0" value="1"/>
</SETPORT>
```

Bei E-Mails und Express-E-Mails sollte zusätzlich der Parameter „Alpha“ (Absenderadresse) geloggt werden.

4.3.6. E-Mails abholen

Damit das Alarm Modem via E-Mail Schaltbefehle erhalten kann, muss dieses zunächst nach E-Mails schauen. Dazu ist ein EventHandler mit dem „POP3Query“-Befehl zu konfigurieren, z.B.:

```
<CheckMail>
  <POP3Query/>
</CheckMail>
```

Dieser EventHandler sollte regelmäßig über den Scheduler aufgerufen werden, oder aber duch einen CallerID-Trigger (Kapitel 4.4) um es mit unserem NewMailSignal-Sevice zu verwenden. Siehe dazu auch: http://www.NewMailSignal.com

4.4. Event durch Anruf (CallerID) H. Kap. 9.2

Das Gerät ist auch in der Lage durch einen simplen Telefonanruf aufgrund der übermittelten Rufnummer einen Event auszuführen, um z.B. einen Ausgang zum Öffnen eines Garagentores zu schalten. Diese Funktion wird in TILA2 ebenfalls angeboten. (`basis-calltrigger.cnf`)

Die Rufnummer und das damit verknüpfte Event (EventHandler) werden auf dem Register `20-Incoming CallerID Alarms` definiert:

```
<IncomingCallTrigger>
     <No1 _="0307654321" event="Switch_0"/>
</IncomingCallTrigger>
```

“`0307654321`” wäre in diesem Fall die übermittelte Rufnummer. Der Parameter „event“ verweist auf den auszuführenden EventHandler, in welchem z.B. der Ausgang gesetzt wird.

Es können auch verschiedene Rufnummer verschiedenen Events zugewiesen werden. Der Parameter „`NoX`“ muss dazu fortlaufend hochgezählt werden.

Mit dem von Tixi kostenlos angebotenen „CLIP-Tool“, können Sie sich das CallerID-Format Ihres Telefonanschlusses ausgeben lassen.

4.5. Projektupload per E-Mail (TiXML-Reference Kapitel 3.8.1)

Wenn man viele ähnliche Anlagen installiert hat, und einen Teil der Konfiguration sich regelmäßig ändert (z.B. das Adressbuch oder Sequenzer-Tabellen), macht eine Projektänderung via E-Mail mehr Sinn, als alle Anlagen nacheinander anzurufen.
Daher ist das Alarm Modem in der Lage, Projektänderungen via E-Mail zu empfangen (`basis-SetConfig.cnf`).

Zunächst richtet man, wie im Kapitel 4.3.6 beschrieben, eine regelmäßige Mailabholung ein.

Über die Betreffzeile der abgeholten E-Mail, welche natürlich wie beim Fernwirken zunächst ein Passwort enthält (Kapitel 4.3.1), wird ein EventHandler mit einer SetConfig-Anweisung aufgerufen (Register 10 – EventHandler):

```
<SETCONFIG>
 <SetConfig/>
</SETCONFIG>
```

In die E-Mail, die an das Alarm Modem geschickt wird, kopiert man nun hintereinander alle SetConfig Anweisungen für die zu ändernden Datenbanken. Die eckigen Klammern des TiXML-Frames sind dabei zu entfernen.
Stattdessen wird der komplette Block von SetConfig-Anweisungen durch ein <D> </D> Tag umschlossen, z.B.:

```
<D>
<SetConfig _="TEMPLATE" ver="y">
<AddressBook>

<MySelf Name="MySelf" hidden="1">
 <SMS_No _="+49-172-6055321"/>
 <Fax _="+49-172-6055321"/>
 <Email _="Tixi-Training1@gmx.net"/>
 <Express-Email _="TAM+49-172-6055321"/>
</MySelf>

<OA hidden="1">
 <SMS_No _="&#xae;~/OA"/>
 <Email _="&#xae;~/OA"/>
 <Express-Email _="&#xae;~/Alpha"/>
 <SMS_Provider _="AnnyWay"/>
</OA>

<Contact_0 Name="Receiver">
 <Email _="demo@tixi.com"/>
 <Express-Email _="TAM+49-30-40608444"/>
 <SMS_No _="+49-172-1844751"/>
 <SMS_Provider _="AnnyWay"/>
 <Pager_No _="394000"/>
 <Pager_Provider _="CityRuf"/>
 <Fax _="+49-30-40608438"/>
</Contact_0>

</AddressBook>
</SetConfig>
</D>
```

Zu beachten ist, dass die E-Mail im E-Mail-Programm im Plain-Text (nur Text) Format erstellt wird. HTML- oder Ritch-Text E-Mails (RTF) werden nicht unterstützt.

5. Anbindung an eine SPS (TiXML SPS Handbuch)

Die bisherigen Beispiele haben als Alarmauslöser oder beim Fernwirken digitale Ein- und Ausgänge des Alarm Modems verwendet.
Anstelle der I/O-Ports können auch SPS-Variablen für alle beschriebenen Funktionen verwendet werden, z.B. für Datenlogging, Alarme und Fernwirken.

In den Beispielen müssen dazu die I/O-Port-Referenzen (z.B. „/Process/MB/IO/I/P0“) auf die SPS-Variablen (z.B. „/Process/Bus1/Device_0/Variable_0“) geändert werden.

Die komplexe Datenbank der SPS-Variablen (Register `17-Externa l`) sollte der Einfachheit halber mit TILA2 erstellt werden.

Das Beispiel `basis-messages-sps.cnf` zeigt exemplarisch eine Alarmfunktion sowie SMS-Fernwirken in Zusammenarbeit mit einer SPS.

6. Weitere Funktionen

Die in diesem Tutorial vorgestellten Funktionen decken den Großteil der vom Anwender geforderten Anwendungen ab.

Für weitere Funktionen und die Verfeinerung der gezeigten Beispiele empfehlen wir das Studium des TiXML Reference Handbuchs, z.B. bei

Logik- und Vergleichsoperationen
Webserver
Datenformatierungen
Scheduler-Conditions
Variablenformatierungen
uvm.

Außerdem empfehlen wir einen regelmäßigen Blick in das Tixi User-Forum auf unserer Webseite (http://www.tixi.com) wo wir und andere Tixi Anwender Programmbeispiele und Tips veröffentlichen.

Quellen:

1 Hans Hubert Partl, Wien, siehe auch http://www.boku.ac.at/htmleinf/xmlkurz.html
2 TiXML Reference Manual, Tixi.Com, 2003