日本語訳 2008 年　10 月

日本語訳作成 (Japanese Translation):

特定非営利活動法人 LonMark Japan

日本語訳編集 (Editors):

田中 宏明	TANAKA, Hiroaki	横河電機株式会社・ LonMark　Japan
高橋 宏雄	TAKAHASHI, Hiroo	株式会社イースト

日本語訳貢献者 (Contributors):

藤原 憲明	FUJIWARA, Noriaki	パナソニック電工株式会社
石山 政浩	ISHIYAMA, Masahiro	株式会社 東芝 研究開発センター
金子 雄	KANEKO, Yu	株式会社 東芝 研究開発センター
前川 智則	MAEGAWA, Tomonori	株式会社 東芝 研究開発センター
山村 晃永	YAMAMURA, Teruhisa	株式会社ＮＴＴファシリティーズ
藤村 文雄	FUJIMURA, Fumio	株式会社山武ビルシステムカンパニー
五島 憲司	GOTO, Kenji	株式会社日本アジルテック
中川 一良	NAKAGAWA, Kazuyoshi	株式会社日本アジルテック
鴨井 亮	KAMOI, Ryo	株式会社日本アジルテック
河村 一	KAWAMURA, Hajime	鹿島建設株式会社
広瀬 啓一	HIROSE, Keiichi	清水建設株式会社

日本語訳にあたり、グリーン東大工学部プロジェクトの多大なる協力をいただきました。

免責事項 (disclaimer notice):

英文文書所在 www.oasis-open.org/committees/download.php/21461/**obix**-**1.0**-cs-01.doc

oBIX 1.0

Committee Specification 01, 5 December 2006

Document identifier:

www.oasis-open.org/committees/download.php/21461/obix-1.0-cs-01.doc

Location:

http://www.oasis-open.org/committees/obix

Technical Committee:

OASIS Open Building Information Exchange TC

Chairs:

Toby Considine, University of North Carolina Chapel Hill

Paul Ehrlich, Building Intelligence Group

Editor:

Brian Frank, Tridium

Abstract:

このドキュメントはオブジェクトモデルとマシン・ツー・マシン（M2M）通信のために使われた XML フォーマットを明示しています。

Status:

この文書は上述の日付をもって oBIX TC によって最新に修正されたか、もしくは承認されました。承認のレベルは同様に記述されます。この文書の後の改訂版がないかどうか上に記された最新のロケーションをチェックしてください。この文書は特定のスケジュールで定期的に更新されるものではありません。

技術委員会のメンバーはこの仕様についてのコメントを技術委員会用の電子メールリストに送ってください。その他の方は以下の技術委員会の Web ページ

www.oasis-open.org/committees/obix にある“Send A Comment”ボタンを利用して技術委員会にコメントを送ってください。この仕様を実装することに不可欠であるかもしれない特許開示情報とライセンス条項の提供については、技術委員会 Web ページ(www.oasis-open.org/committees/obix/ipr.php)の知的財産権の項を参照してください。

この仕様のための非標準の正誤表ページは www.oasis-open.org/committees/obix にあります。

Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an “AS IS” basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS は、本文書で記述された技術の実装や利用に関して主張される可能性がある知的財産や他の権利の正当性や範囲についてや、そのような権利のライセンスが利用可能または不可能かもしれないことについて、何の立場もとらないし、そのような権利を確認するために努力してきたとも主張しない。OASIS 仕様の権利に関する OASIS の手続き情報は OASIS ウェブサイトで見ることができる。公表のために利用可能となっている権利の主張の写しと利用可能となるであろうライセンスの保証、または、本仕様の実装者または利用者がそのような財産権を利用するための一般的なライセンスまたは許可を取得しようとした試みの結果は、OASIS Executive Director から取得することができる。

OASIS は、関心のあるものは誰でも、本仕様を実装するために必要とされる技術を対象とする著作権、特許または特許申請、または、他の財産権についての注意をうながしてもらうようお願いする。このような情報については、OASIS Executive Director に連絡してほしい。

もよい。しかしながら、OASIS 仕様を開発するという目的のために必要とされる場合 (この場合、OASIS IntellectualProperty Rights 文書中で定義される著作権のための手続きにしたがわなければならない) や英語以外の言語へ翻訳するために必要とされる場合を除いて、本文書自身は著作権表示または OASIS への参照を削除するなどどのような方法でも改変できない。上で付与した制限付きの許可は永続的なものであり、OASIS もしくはその後継者または任命者によって取り消されることはない。本文書およびここに含まれる情報は「現状有姿」のままで提供され、この情報の利用がどのような権利も侵害しないという保証や、商品性や特定の目的への適応性についての暗黙の保証を含むがこれらに限らず、明示的または暗示的を問わず、一切の保証を **OASIS** は行なわない。

本節は参考である。

Table of Contents

261

1 Overview

oBIX は我々を取り巻く世界中の計測と制御を行う組み込みソフトウェアへのアクセスを提供するように設計されています。歴史的にはこれらをシステムに統合するには専用の低レベルのプロトコルを必要とし、またしばしば専用の物理的なネットワークインタフェースを必要としました。しかし現在、急激に増加するユビキタスネットワークと低価格の組み込み機器用の高性能なマイクロプロセッサが利用できることが、インターネットのまさしくその構造の中にこれらのシステムを作り上げています。一般的にマシン対マシンを意味する用語 M2M はインターネットの世界で完全な変化を起こしており、Web とマシンとの自律的な相互通信への進展という新しいページを開いているのです。oBIX 仕様は M2M Web が利用する XML、HTTP、URI といった標準的で企業に認知された技術を利用することに基礎を置いて構築されています。

以下の設計ポイントは oBIX が解決しようと試みる問題を説明します:

- **XML**: M2M 情報を標準的な XML 構文で表します。
- **Networking**: ネットワークを介して XML による M2M 情報を転送します。
- **Normalization**: M2M の共通な特徴のための標準表現**:**points, histories, alarms;
- **Foundation**: 新しい標準に共通のカーネルを提供します。

1.1 XML

oBIX の第一要件は多様な M2M システムからの情報を表す共通の XML 構文を開発することです。oBIX の設計思想はシンプルに定義された XML 構文 にマップする小さいが拡張可能なデータモデルに基づいています。このコアオブジェクトモデルとその XML 構文は４章に掲載されている１つのイラストに十分簡素に表現されています。オブジェクトモデルの拡張性はコントラクトと呼ばれる概念を通して新しい抽象概念の定義を可能にします。oBIX 仕様の大部分は実際にコントラクトを通して oBIX それ自身で定義されます。

1.2 Networking

我々が M2M 情報を XML で表す方法を持つと同時に次のステップは公開と活用のためにネットワークを介してそれを転送するために標準的な仕組みを提供することです。

oBIX はネットワークを２つに分割します: 抽象的なリクエスト / レスポンスモデルとそのモデルを実行する一連のプロトコルバインディングです。oBIX のバージョン１.０は既存の Web サービスの基盤を導入するように設計された２つのプロトコルバインディングを定義します: HTTP REST バインディングと SOAP バインディングです。

1.3 Normalization

実世界を計測し制御するシステムの中に広い適用性を持つ幾つかの概念があります。oBIX のバージョン１.０はこれらの３つに標準化された表現を提供します:

- **Points**: 1 つの計測値とその状態を表す – 一般的にこれらはセンサー、 アクチュエーター 、あるいはセットポイントのようなコンフィギュレーション変数にマップされます;

- **Histories**: 時系列にサンプリングされたポイントデータのモデリングと問い合わせです。一般に端末装置は分析のためのより高度なアプリケーションに入力されるタイムスタンプ付きのポイントデータを集めます。
- **Alarming**: アラームのモデリング、ルーティング、確認です。アラームは、ユーザあるいは他のアプリケーションへの通知を必要とする条件を示します。

1.4 Foundation

M2M システムのための要求項目と縦に連なった複数ドメインの問題は非常に広範囲です– １つの仕様でカバーするにはあまりにも広範過ぎます。

oBIX は明らかに低レベルの仕様ですが、しかしコントラクトに基づいた強力な拡張メカニズムを備えて計画的に設計されています。oBIX のゴールは新しい仕様のために基礎を提供する共通のオブジェクトモデルと XML 構文のための整備をすることです。

縦に連なった複数ドメインのための仕様の積み重ねが oBIX の上に共通の規約として構築されることが望まれます。

2 Quick Start

この章は即座に oBIX に取り組み、全てのタグの習得を望む熱心な読者のためのものです。始めるのによい方法は誰でもが精通している単純な例 – 安定したサーモスタットを例にすることです。我々が非常に単純なサーモスタットを持っていると仮定しましょう。それは現在の室内温度を通知する温度センサーを持っています、そしてそれは決められた温度を維持するセットポイントを持っています。我々のサーモスタットがただ暖房動作をするだけであると仮定しましょう、そしてサーモスタットは暖房装置が現在動作するべきであるかどうか通知するデータを持っています。我々のサーモスタットが oBIX XML でどう見えるかひと目見ましょう：

```
<obj href="http://myhome/thermostat">
  <real name="spaceTemp" units="obix:units/fahrenheit" val="67.2"/>
  <real name="setpoint" unit="obix:units/fahrenheit" val="72.0"/>
  <bool name="furnaceOn" val="true"/>
</obj>
```

最初に注意するべきことは３つの要素タイプがあるということです。

oBIX ではオブジェクトと要素の間に１対１のマッピングがあります。

オブジェクトは oBIX データモデルによって使われる基本的な抽象概念です。

要素は XML 構文でそれらのオブジェクトがどのように表されるかを示します。

このドキュメントは用語としてオブジェクトとサブオブジェクトを使います – しかし、XML 表現について話をするときには、用語として、要素とサブ要素を代わりに利用できます。

ルート obj 要素は全部のサーモスタットをモデル化します。その href 属性はこの oBIX ドキュメントに対して URI を識別子とします。サーモスタットの各変数のために３つの子オブジェクトがあります。real オブジェクトは２つの浮動小数点値を保持します：室内温度と設定ポイントです。bool オブジェクトは暖房装置の状態のために boolean 変数を保持します。それぞれのサブ要素が親要素の中の役割を明確にする name 属性を含みます。それぞれのサブ要素が同様に現在の値のために val 属性を含んでいます。最後に、我々が摂氏ではなく、華氏であることを知っている通り units と呼ばれる属性で温度に注釈がつけられていることがわかります（それは暑い部屋でしょう）。oBIX 仕様はひと塊のファセットと呼ばれるこれらの注釈を定義しています。

現実には、センサーとアクチュエーター 変数（ポイントと呼ばれる）が単純な計測値よりも多くのセマンティクスを示します。アラームのような他のケースでは、複雑なデータ構造を標準化することは望ましいことです。oBIX はコントラクトの中にこれらの概念を取り込みます。コントラクトが我々に標準化されたセマンティクスと構造でオブジェクトにタグを付けることを許します。

サーモスタットのセンサーが華氏-412 の値を示していることを想定してください。

明らかに我々のサーモスタットは壊されています、それで我々はフォールト状態を報告するべきです。XML を書き直して status ファセットを含ませ、そしてコントラクトを使用して追加のセマンティクスを提供しましょう:

```
<obj href="http://myhome/thermostat/">

  <!-- spaceTemp point -->
  <real name="spaceTemp" is="obix:Point"
        val="-412.0" status="fault"
        units="obix:units/fahrenheit"/>

  <!-- setpoint point -->
  <real name="setpoint" is="obix:Point"
        val="72.0"
        unit="obix:units/fahrenheit"/>

  <!-- furnaceOn point -->
  <bool name="furnaceOn" is="obix:Point" val="true"/>

</obj>
```

３つの計測値が is 属性を通して obix:Points としてタグ付けされていることに注意してください。これは標準化されたポイント情報を表すために oBIX によって定義される標準的なコントラクトです。これらのコントラクトを実装することによって、クライアントはすぐに意味的にこれらのオブジェクトをポイントとして取り扱うことを知ります。

コントラクトは oBIX における新しい抽象概念をコアとなるオブジェクトモデル上に構築するための重要な役割を果たします。コントラクトは巧みです、何故ならそれらは標準的な oBIX 構文を使って定義される普通のオブジェクトに過ぎません（ポイント用のコントラクトについてポイント 13 章を参照して下さい）。

3 Architecture

oBIX アーキテクチャは、以下の原理に基づいています。

- **Object Model:** すべての oBIX 情報を定義するのに使われる簡便なオブジェクトモデル。
- **XML Syntax:** オブジェクトモデルを表現する単純な XML 構文です。
- **URI**: URI は、オブジェクトモデルで情報を識別するのに使われます。
- **REST**: 動詞のセット（HTTP GET,PUT,POST）、URI を使ってオブジェクトにアクセスしたり、XML で状態を転送するのに使われます。
- **Contracts:** 新しい oBIX タイプを表すためのテンプレートモデルです。
- **Extendibility:** これらのコンセプトだけを使って一貫した拡張性を提供します。

3.1 Object Model

oBIX 上のすべての情報は、小さな、固定のプリミティブセットを使って表現されます。これらのプリミティブに対するベース抽象化は、オブジェクトと呼ばれます。ひとつのオブジェクトは URI で指定され、すべてのオブジェクトは他のオブジェクトを含むことができます。

単純な情報を保持するために使われるバリューオブジェクトの 8 個の特別な種別があります:

- *bool*: ブール値を保持します - true または false;
- *int*: 整数値を保持します;
- *real*: 浮動小数点値を保持します;
- *str*: UNICODE 文字列を保持します;
- *enum*: 固定レンジの列挙型を保持します;
- *abstime*: 絶対時間を保持します (タイムスタンプ);
- *reltime*: 相対時間を保持します (期間または、時間間隔);
- *uri*: URI を保持します;

どのバリューオブジェクトもサブオブジェクトを含むことができることに注意してください。他に特別なオブジェクトタイプとして、list, op, feed, ref, err　があります。

3.2 XML

oBIX は、その基礎をなしているオブジェクトモデルを表現するための簡単な XML 構文です。各オブジェクトタイプは、要素の 1 タイプに指定されます。値オブジェクトは、val 属性を使ってデータを表します。その他全ての集合は、単純にタグを重ねていきます。単純な例は以下のようになります:

```
<obj href="http://bradybunch/people/Mike-Brady/">
  <obj name="fullName">
    <str name="first" val="Mike"/>
    <str name="last" val="Brady"/>
  </obj>
  <int name ="age" val="45"/>
  <ref name="spouse" href="/people/Carol-Brady"/>
```

```
  <list name="children">
    <ref href="/people/Greg-Brady"/>
    <ref href="/people/Peter-Brady"/>
    <ref href="/people/Bobby-Brady"/>
    <ref href="/people/Marsha-Brady"/>
    <ref href="/people/Jan-Brady"/>
    <ref href="/people/Cindy-Brady"/>
  </list>
</obj>
```

この単純な例では、オブジェクトについてのさらなる情報を得るために使われるかもしれない href 属性がどのように URI リファレンスを指定しているかに注意してください。

name と href の詳細は、5 章で論ぜられます。

3.3 URIs

アーキテクチャを完全なものとするためにある種のネーミングシステムが必要です。oBIX では全てがオブジェクトです、そこでオブジェクトに名前を付ける方法が必要です。

oBIX が XML を使用して web 上に情報を生成しており、さらに URI(Uniform Resource Identifier)を利用することに意味があります。 URI は web で ”リソース ” を識別する標準的な方法です。

URI は、同様にしばしばどのようにそれらのリソースを取り出すべきかについての情報を提供します ― それはそれらがしばしば URL(Uniform Resource Locator) と呼ばれる理由です。

実際的な見地から、もしベンダーが HTTP URI をそれらのオブジェクトを識別するために使うなら、あなたはそのオブジェクトのために oBIX ドキュメントを取り出すために多くの場合シンプルな HTTP GET を行うことができます。しかし技術的には、 URI の内容を取って来ることは後の章で論ずるバインドプロトコルの問題です。

URI の価値はそれらがすでに我々のために定義されたあらゆる種類の気がきいたルールを持っているということです（RFC 3986 参照）。

例えば URI はどの文字が正当であるか、そしてどの文字が不当であるか定義しています。

oBIX の重要な価値は URI リファレンスであり、それは相対的な URI の表現と標準化の方法を定義します。さらにたいていのプログラミング環境は URI を管理するためにライブラリを持っています、したがって、開発者は標準化の詳細の核心について心配する必要がありません。

3.4 REST

多くの経験豊富な読者が URI によって識別され、XML ドキュメントとして転送されるオブジェクトは REST だと考えているかもしれません – それは正しいです。REST は [REpresentational State Transfer](具象的な状態の転送)を表し、そして、REST は World Wide Web の機能に類似した Web サービスの構築スタイルです。WWW は基本的に URI を利用してすべて一緒にハイパーリンクされた HTML ドキュメントの大きなクモの巣です。同じく、 oBIX は基本的に URI を使って相互にハイパーリンクされた XML オブジェクトドキュメントの大きなウェブ（クモの巣）です。

REST は仕様よりも、デザインスタイルです。REST は、メソッド重点主義ではなくリソース重点主義です- リソースは oBIX オブジェクトです。全てのリソースと共に使われる動詞の数は、少なくなる傾向があります。oBIX ではすべてのネットワークリクエストは３つのリクエストタイプに要約されます:

- **Read**: an object　　オブジェクトの読み込みです。
- **Write**: an object　　オブジェクト書き込みです。
- **Invoke**: an operation　　(オブジェクトの)操作です。

3.5 Contracts

すべてのソフトウェアドメインでは、多くの異なったオブジェクトのインスタンスが共通の特徴を共有する場合、パターンが発生し始めます。

例えば人々をモデル化して扱うたいていのシステムは、それぞれの人がおそらく名前、アドレス、電話番号を持っています。縦に連なった複数のドメインで我々はドメイン特定の情報をそれぞれの人に付与するかもしれません。例えばアクセス制御システムが ID 番号をそれぞれの人と結び付けるかもしれません。

オブジェクト指向システムでは我々はクラスの中にこれらのパターンを取り込みます。

リレーショナルデータベースでは我々は型を決められた列でそれらをテーブルに設定します。

oBIX で我々はコントラクトと呼ばれる概念を使ってこれらのパターンを取り込みます、そしてそれはテンプレートとして使用される標準的な oBIX オブジェクトです。コントラクトは、新しい構文を導入することについてのオーバーヘッドがないという点で、厳密に型を決められたスキーマ言語より融通性があり、柔軟です。コントラクトドキュメントが他のいかなる oBIX ドキュメントとでもまったく同じように解析されます。技術オタクの言葉でコントラクトはプロトタイプベースの継承と mixin の組み合わせです。

我々はなぜこれらのパターンを取り込もうとすることに関心を持つのでしょうか？

コントラクトの最も重要な使用は oBIX 仕様自身によって新しい標準的な抽象概念を定義することです。標準化されたセマンティクスについて合意することは皆が構文についてそうするのと同じぐらい重要です。コントラクトが OO 男性クラスやリレーショナルデータベースの男性テーブルをマップするのに必要な定義を提供します。

3.6 Extendibility

我々は oBIX を縦に連なった複数のドメインで新しい抽象概念を開発するための基盤として使用することを望みます。　我々はまた、旧式なシステムと新しいプロダクトラインにまたがって oBIX を実装しようとするベンダーに対して拡張性を提供することを望みます。さらに、それは白紙の状態で出荷され、現場にて完全にプログラミングされるデバイスに共通です。これは我々に標準ベースの、ベンダーベースの、そしてさらにプロジェクトベースの拡張性を残しています。

oBIX の拡張性 の背後の原則はあらゆる新しいものがオブジェクト、URI、コントラクトを使って厳密に定義されるということです。別の言い方をすれば ― 新しい抽象概念は、クライアントコードが注意することを強いられる新しい XML 構文あるいは機能を導入しません。新しい抽象概念が常にただ異なったセマンティクスで oBIX オブジェクトの標準的なツリーとしてモデル化されます。それはもっと高レベルのアプリケーションコードが

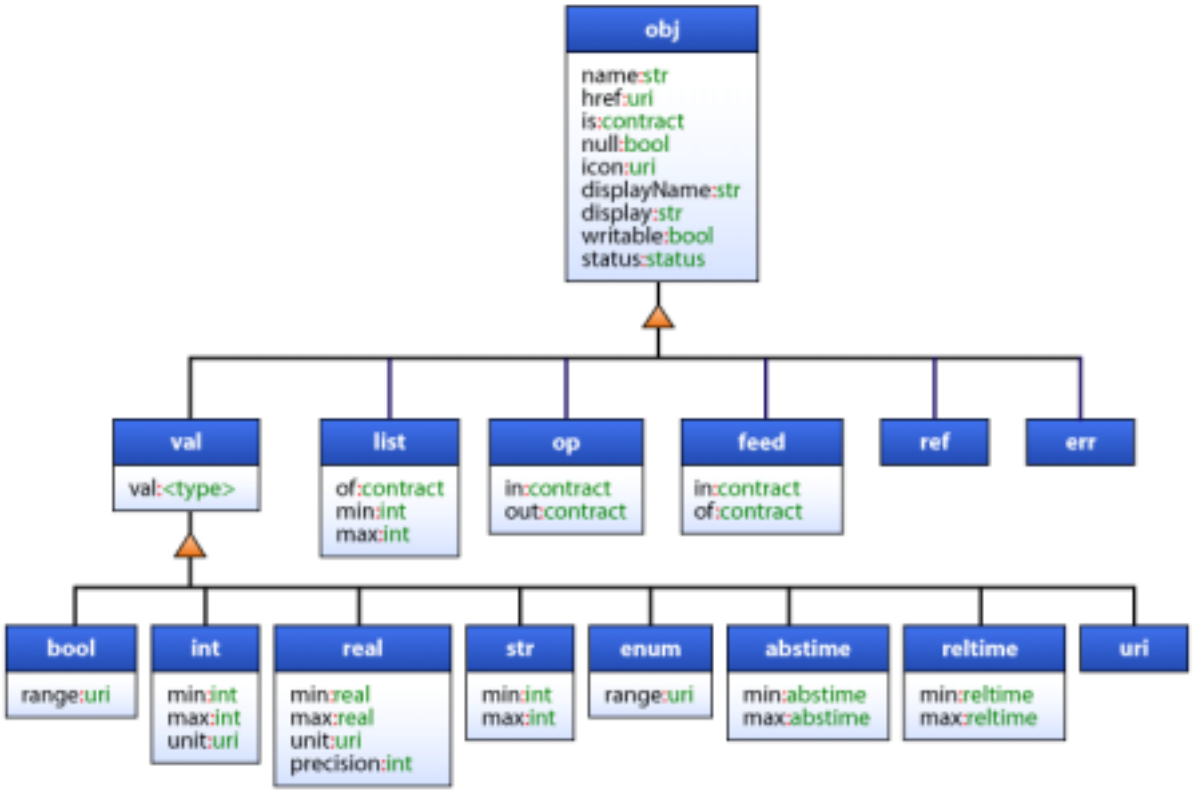
新しい抽象概念を取り扱うために決して変わらないということを意味しません、しかしネットワークと構文解析を扱うコアスタックが変化するべきではありません。

この 拡張 モデルは Java あるいはＣ＃のようなほとんどの主流のプログラム言語に類似しています。中核となる言語の構文は新しい抽象概念を定義するために組み込みのメカニズムで決められます。

拡張性は言語の固定された構文を使って新しいクラス・ライブラリの定義によって達成されます。これは、誰かが新しいクラスを加える時いつも、コンパイラを更新しなくてもよいことを意味します。

4 Object Model

oBIX 仕様は小さな、決められたオブジェクト型のセットに基づいています。オブジェクト型は１対１で XML 要素型に対応します。oBIX オブジェクトモデルは次の図で要約されます。それぞれの箱が特定のオブジェクト型（XML 要素名）を表します。それぞれのオブジェクト型は、オブジェクトがサポートしている属性も示しています。



4.1 obj

oBIX の抽象化の基本はオブジェクトであり、obj 要素の XML でモデル化されています。oBIX の全ての XML 要素は obj 要素から派生します。任意の obj 要素またはその派生要素は他の obj 要素を包含することができます。obj 要素のサポートする属性は以下を含みます:

- **name**:親オブジェクト内でのオブジェクトの目的を定義します（５章で扱います）；
- **href**:オブジェクトを識別する URI 参照を付加します（５章で扱います）；
- **is**:オブジェクトが実装するコントラクトを定義します（６章で扱います）；
- **null**: **null** オブジェクトをサポートします(セクション 4.15 で扱います)
- **facets**: オブジェクトのメタデータが付加される属性のセットです(セクション 4.156 で扱います)

- **val**:値オブジェクトのみで使用する実際の値を格納する属性です(bool, int, real, str, enum, abstime, reltime, uri)。

obj のコントラクト定義は次のようになります:

```
<obj href="obix:obj" null="false" writable="false" status="ok" />
```

4.2 bool

bool オブジェクトは true または false の論理条件を表します。val 属性は xs:boolean で指定し、デフォルトは false です。bool のリテラル値は“true” または “false”です(リテラルとして “1” と “0” は許可されていません)。コントラクト定義:

```
<bool href="obix:bool" is="obix:obj" val="false" null="false"/>
```

例:

```
<bool val="true"/>
```

4.3 int

int 型は整数を表します。val 属性は xs:long で指定し、64 ビットの整数でデフォルトは 0 です。コントラクト定義:

```
<int href="obix:int" is="obix:obj" val"0" null="false"/>
```

例:

```
<int val="52"/>
```

4.4 real

real 型は浮動小数点数を表します。val 属性は xs:double で指定し、IEEE64 ビット浮動小数点数であり、デフォルトは 0 です。コントラクト定義:

```
<real href="obix:real" is="obix:obj" val="0" null="false"/>
```

例:

```
<real val="41.06"/>
```

4.5 str

str 型は Unicode 文字列を表します。val 属性は xs:string で指定し、デフォルトは空文字列です。コントラクト定義:

```
<str href="obix:str" is="obix:obj" val="" null="false"/>
```

例:

```
<str val="hello world"/>
```

4.6 enum

enum 型は有限要素集合の一要素に一致する値を表すために使用されます。有限の値の組は範囲(range)と呼称されます。enum の val 属性は xs:string を用いて文字列キーとして表現されます。enum のデフォルトは null です。enum の範囲は range 属性を用いてファセットで宣言されます。コントラクト定義:

```
<enum href="obix:enum" is="obix:obj" val="" null="true"/>
```

例:

```
<enum range="/enums/OffSlowFast" val="slow"/>
```

4.7 abstime

abstime 型は絶対日時を表すために利用されます。val 属性は xs:dateTime で指定します。デフォルトでは null に設定されます。コントラクト定義:

```
<abstime href="obix:abstime" is="obix:obj" val="1970-01-01T00:00" null="true"/>
```

2005 年 3 月 9 日 午後 1 時 30 分 GMT(グリニッジ標準時)の例:

```
<abstime val="2005-03-09T13:30Z"/>
```

4.8 reltime

reltime 型は相対的な経過時間を表すために利用されます。val 属性は xs:duration で指定され、デフォルトは 0 秒です。コントラクト定義:

```
<reltime href="obix:reltime" is="obix:obj" val="PT0S" null="false"/>
```

15 秒の例:

```
<reltime val="PT15S"/>
```

4.9 uri

uri 型は URI 参照を格納するために利用されます。通常使われている str と異なり、uri は RFC3986 と XML スキーマの anyURI 型によって定義されているような限定された字句空間をもっています。多くの URI は URL でもあり、リソースを特定し（主に HTTP によって）リソースを検索するために使われます。コントラクト定義:

```
<uri href="obix:uri" is="obix:obj" val="" null="false"/>
```

oBIX のホームページ例:

```
<uri val="http://obix.org/" />
```

4.10 list

list オブジェクトは他のオブジェクトのリストを格納するための専用のオブジェクト型です。一般的な obj に対して list オブジェクトを利用する主な利点は、of 属性を使うことでコンテンツの共通のコントラクトを指定することができる点です。もし指定されたなら、コントラクトリストでフォーマットされた URI のリストは、of 属性で指定された要素で構成されていなくてはなりません。

リストの定義の例:

```
<list href="obix:list" is="obix:obj" of="obix:obj"/>
```

リスト文字列の例:

```
<list of="obix:str">
  <str val="one"/>
  <str val="two"/>
</list>
```

リストはセクション 6.8 でコントラクトとともにより詳細に論じられます。

4.11 ref

ref オブジェクトは他の oBIX オブジェクトへの参照を定義するために使用されます。これは HTML のアンカータグ(<a>)と oBIX では同じ役割です。コントラクト定義:

```
<ref href="obix:ref " is="obix:obj"/>
```

ref 要素は常に href 属性を指定しなくてはなりません。参考事項がセクション 9.2 で詳細に論じられます。

4.12 err

err オブジェクトはエラーを示すために使われる特別なオブジェクトです。その実際のセマンティクスは文脈に依存します。一般に err オブジェクトは display 属性によって問題点について人間が読むことができる記述で表現するべきです。コントラクト定義：

```
<err href="obix:err" is="obix:obj"/>
```

4.13 op

op オブジェクトは操作を定義するために使われます。すべての操作は 1 つのインプットオブジェクトをパラメータとし、アウトプットとして 1 つのオブジェクトを返します。インプットコントラクトとアウトプットコントラクトは in と out 属性により定義されます。コントラクト定義：

```
<op href="obix:op" is="obix:obj" in="obix:Nil" out="obix:Nil"/>
```

オペレーション（操作）は 8 章で詳細に論じられます。

4.14 feed

feed オブジェクトはイベントを送るためのトピックを定義するために使われます。

feed はアラームのようなイベントの流れを定期受信し監視するために使われます。

feed は of 属性で通知されるイベント型を指定するべきです。in 属性は feed への定期受信を行うときにインプット引数を渡すために使うことができます（フィルタの例のように）。

```
<feed href="obix:feed" is="obix:obj" in="obix:Nil" of="obix:obj"/>
```

watch を通して定期受信される feed が 12 章で論じられます。

4.15 Null

全てのオブジェクトは null 概念をサポートしています。Null は値のない状態です。Null は論理値の null 属性を使用して示されます。enum と abstime は除いた全てのオブジェクトは null の デフォルトに false を設定しています(他のいかなるデフォルトでも紛らわしいために)。Null は他の属性より少し異なってコントラクトから継承されます。詳細についてはセクション 6.4 を参照ください。

4.16 Facets

すべてのオブジェクトはファセットと呼ばれる所定の属性を使うことで注釈を付けることができます。ファセットはオブジェクトについて追加のメタデータを付加します。利用可能なファセットは： displayName, display, icon, min, max, precision, range, unit です。ベンダーがファセットを追加しオブジェクトに注釈を付ける場合、属性を限定する XML 名前空間を使うことを検討すべきです。

4.16.1 displayName

displayName ファセットは xs:string として設定され、人が読むことができるローカライズされたオブジェクトの名前を付加します:

```
<obj name="spaceTemp" displayName="Space Temperature"/>
```

一般に displayName ファセットは名前属性のローカライズされた書式であるべきです。

displayName へのコントラクトからのオーバライドには制限はありません（displayName は単に name のヒューマンフレンドリーなバージョンなので、一般的ではないでしょう）。

4.16.2 display

display ファセットは人が読むことができるローカライズされたオブジェクトの説明を提供し、xs:string で設定します:

```
<bool name="occupied" val="false" display="Unoccupied"/>
```

display へのコントラクトからのオーバライドには制限はありません。

display 属性は Java あるいはＣ # で Object.toString () と同じ目的を果たします。それはすべてのオブジェクトに文字列表示を指定する一般的な方法を提供します。

value オブジェクト(bool / int)の場合にはローカライズされ書式を合わせた val 属性の表現を提供すべきです。

4.16.3 icon

icon ファセットはユーザーエージェント内でオブジェクトを表示するために利用するかもしれないグラフィックアイコンの URI 参照を提供します:

```
<object icon="/icons/equipment.png"/>
```

icon 属性の内容はイメージファイルへの URI でなければなりません。イメージファイルはなるべく 16x16 PNG をみたす ファイルであるべきです。icon へのコントラクトからのオーバライドには制限はありません。

4.16.4 min

min ファセットはその値を含んだ最小値を定義するために使われます:

```
<int min="5" val="6"/>
```

min 属性の内容はその関連づけられた val 型と一致しなくてはなりません。

min ファセットは int, real , abstime, reltime のその値を含んだ下限値を定義するために使われます。それは str では文字列の Unicode 文字の最小数を表すために使われます。

リストでは（名前付き、あるいは名前なしの）子オブジェクトの最小数を示すために使われます。min ファセットのオーバライドはより大きい値を使って値域を狭めることができます。min ファセットは、（等値であることはできますが）決して max ファセットより大きくしてはなりません。

4.16.5 max

max ファセットはその値を含んだ最大の値を定義するために使われます:

```
<real max="70" val="65"/>
```

max 属性の内容はその関連づけられた val 型と一致しなくてはなりません。

max ファセットは int, real, abstime, reltime では、その値を含んだ上限値を定義するために使用されます。str では文字列の Unicode 文字の最大数を示すために使われます。list では、（名前有りまたは名前無しの）子オブジェクトの最大値を示すために使われます。max ファセットのオーバライドはより小さい値で値域を狭めるかもしれません。max ファセットは、（等値であることはできますが)min ファセットよりも小さくなることはできません。

4.16.6 precision

precision ファセットは real 値に使われ、小数点の桁を記述するために使われます:

```
<real precision="2" val="75.04"/>
```

692 precision 属性の内容は xs:int でなくてはなりません。precision 属性の値は有意な小数位の
693 桁数と同じです。上記の例で、値が 2 というのは 2 桁の有意な少数部を示します:
694 “ 75.04 ”。一般に precision は real 値の書式設定を行うクライアントアプリケーションに
695 よって使われます。precision へのオーバライドには制限はありません。

696 ### 4.16.7 range

697 range ファセットは列挙の値域を定義するために使われます。range 属性は obix:Range オ
698 ブジェクトへの URI 参照です（定義のために 11.2 を参照）。 bool と enum オブジェクト
699 型で使われます:

700 ```
<enum range="/enums/OffSlowFast" val="slow"/>
```

701 range においてのオーバライドルールは指定された範囲がコントラクトの範囲から継承さ
702 れなくてはならないということです。enum の特殊性として、通常範囲に新しい項目を加
703 えることを伴うという点で列挙は面白い性質を持っています。技術的には、これはそれを
704 狭めるよりむしろ enum の値域を広げています。けれども実際は、範囲の中に項目を加え
705 ることは我々が切望するものです。

706 ### 4.16.8 status

707 status ファセットは情報の品質と状態についてオブジェクトに注釈を付けるために使われ
708 ます:

709 ```
<real val="67.2" status="alarm"/>
```

710 ステータスは（優先順に並べられた）以下の値の１つを持っている列挙された文字列値で
711 す:

712 - **disabled**: この状態（ステータス）はオブジェクトが正常なオペレーションから
713 disable（OutOfService）にされたことを示します。オペレーションと feed に関
714 しては、この状態はオペレーションあるいは feed に対するサポートを停止するた
715 めに使われます。
716 - **fault**: fault 状態はデータが失敗状態のために無効であるか、あるいは利用できな
717 いことを示します---データが古い、コンフィギュレーション問題、ソフトウェア
718 失敗、あるいはハードウェア故障が該当します。通信を含めての失敗は down 状
719 態を使うべきです。
720 - **down**: down 状態は通信の失敗を示します。
721 - **unackedAlarm:** unackedAlarm 状態はユーザによって確認されていない警報状態
722 があることを示します --- それは、alarm と unacked 状態が重なっている状態で
723 す。alarm と unackedAlarm の違いは、alarm がユーザによって確認済みであるか、
724 あるいは人が確認を行わなくても良い警報状態であることを意味しています。
725 unackedAlarm と unacked の間の相違は、unacked では、オブジェクトが正常な
726 状態に戻ったということです。
727 - **alarm**: この状態はオブジェクトが現在警報状態にあることを示します。アラーム
728 状態は一般にオブジェクトがその正常な境界線外で稼働していることを意味しま
729 す。アナログポイントに関してはこれは現在の価値がその設定された限界の上に
730 あるいは下であることを意味するかもしれません。あるいはそれはデジタルのセ
731 ンサーが望まれていない状態に移行したことを意味するかもしれません。追加の
732 情報のために Alarming（１５章）を見てください。
733 - **unacked:** unacked 状態は未確認のままの過去の発生警報を示すために使われま
734 す。

- **overridden**: overridden 状態は、データは OK だが 、ローカルなオーバライドが現在有効である状態です。オーバライドの例としては、通常のスケジュールの設定ポイントを一時的に書き換えを行うようなことです。
- **ok:** ok 状態は正常な状態を示します。全てのオブジェクトはデフォルトとして ok が仮定されます。

ステータスは上記の列挙された文字列の１つでなければなりません。同時に多数のステータス状態を示すことは固有のシステムで可能であるかもしれません、しかしながら oBIX で設定する場合には、優先順位の高い状態が選択されるべきです。 --- 優先順位は最上位（disabled）から最下位（ok）の順番です。

4.16.9 unit

unit ファセットは測定の単位を定義します。unit 属性は obix:Unit オブジェクトの URI 参照です（コントラクト定義のためにセクション 11.5 を参照）。それは int と real オブジェクト型で使われます:

```
<real unit="obix:units/fahrenheit" val="67.2"/>
```

もしコントラクト内で宣言されるなら、unit ファセットはオーバライドされないことをお勧めします。もしそれがオーバライドされるならば、オーバライドはコントラクトと同じ次元でユニットオブジェクトを使うべきです（それが同様の物理量を測らなくてはなりません）。

4.16.10 writable

writable ファセットはこのオブジェクトがクライアントによって書き込み可能かどうかを明示します。もし false であるなら（デフォルト）、オブジェクトは読み取り専用です。

それはオペレーションと feed 以外すべてのオブジェクトで使われます：

```
<str name="userName" val="jsmith"     writable="false"/>
<str name="fullName" val="John Smith" writable="true"/>
```

5 Naming

すべての oBIX オブジェクトは２つの潜在的な識別子を持っています：name と href です。name はその親の（上位表現）内でオブジェクトの役割を定義するために使われます。name はプログラムの識別子です； displayName ファセットは人との対話のために使われるべきです。最初の文字を小文字にするキャメル記法を使用します。name の主要な目的はサブオブジェクトに意味を付けることです。name はまた、コントラクトからのオーバライドを表示するためにも使用します。名前の良い類似は Java または C#でのクラスのフィールド/メソッド名です。href は URI をオブジェクトに関連付けるために使われます。href は常に URI の参照であり、相対的な URI はベースとなる URI に対して正規化を必要とするかもしれないことを意味します。このルールの例外は oBIX ドキュメント内におけるルートオブジェクトの href です---この href は絶対的な URI とならねばならず、URI 参照ではありません。これはルートオブジェクトの href が正規化のために有効なベース URI（xml:base）として使用されることを示します。共通点のある良い例として HTML や XLink における href があります。

いくつかのオブジェクトが name と href 、name のみ、href のみを持っているかもしれず、あるいは両方とも持っていないかもしれません。リスト内のオブジェクトにとって name を使わないことは普通のことなので多くのリストは name なしのオブジェクトの連続です。oBIX 仕様は name と href を明確に区別します ― あなたは name と href の間の関係を仮定するべきではありません。

実際的な見地から多くのベンダーが多分名前構造をまねて href 構造を構築するでしょう、しかしクライアントソフトウェアは決してこのような関係を想定するべきではありません。

5.1 Name

オブジェクトの name は name 属性を使って表されます。

name はそれらの有効な文字セット上の制約を持っているプログラム識別子です。

name は ASCII 文字、数字、アンダーバー、ドル記号だけを含んでいなくてはなりません。

数字は最初の文字としては使用してはいけません。規約では最初の文字を小文字としたキャメル記法を使用します: “ foo”, “fooBar”, “thisIsOneLongName”。既知のオブジェクトの中で、その直接の子オブジェクトのすべてがユニークな名前を持っていなくてはなりません。name 属性を持っていないオブジェクトが名前なしオブジェクトと呼ばれます。oBIX ドキュメントのルートオブジェクトは name 属性を指定するべきではありません（しかしほとんど常に絶対的な href URI を持ちます）。

5.2 Href

オブジェクトの href は href 属性を使って表されます。もし明示されるなら、ルートオブジェクトは絶対的な URI を持っていなくてはなりません。oBIX ドキュメントの中のすべての（ルートではない）他の href は URI のリファレンスとして扱われ、それらは相対的であるかもしれません。なぜならルート href は常に絶対的な URI であり、それはドキュメント内で標準化された相対的な URI へのベースとして使用されるかもしれないからです。URI 構文のための正式な規則と標準化は RFC 3986 で定義されます。oBIX 内セクション 5.3 のデザインパターンとして提供されるいくつかの共通例を考えます。

一般的な規則として、すべての読み取り可能なオブジェクトに対しては URI を指定しなくてはなりません。読み取りリクエストによって返される oBIX ドキュメントにはルート URI を指定しなくてはなりません。しかしながら、操作によっては算出されたオブジェクトのようにオブジェクトが一時的であるいくつかの例があります。これらの場合、ルート URI がないかもしれません。それは、再びこの特定のオブジェクトを検索する方法がないことを意味します。もしルート URI が提供されないなら、サーバのオーソリティ URI が相対的な URI リファレンス解決のためにベース URI とされます。

5.3 HTTP Relative URIs

よく定義され標準化されたセマンティクスを持っているため、HTTP URI を使用することを推薦しますが、ベンダーはどんな URI 体系でも使うことは自由です。

このセクションでは HTTP URI の標準化が oBIX クライアントエージェントの中でどのように機能するべきであるかの要約を提供します。一般的な規則は:

- もし URI が“*scheme*:”から始まるなら、それは世界的規模で絶対的な URI です。
- もし URI が一つのスラッシュから始まるなら、それはサーバの絶対的な URI です。
- もし URI が“#”から始まるなら、それは（次のセクションで論じられる）フラグメント識別子です。
- もし URI が“../”から始まるなら、パスはベースからの上位階層でなければなりません。

さもなければ URI はベース URI からの相対的なパスであると考えられます。

いくつかの例:

```
http://server/a    +  http://overthere/x  →  http://overthere/x
http://server/a    +  /x/y/z              →  http://server/x/y/z
http://server/a/b  +  c                   →  http://server/a/c
http://server/a/b/ +  c                   →  http://server/a/b/c
http://server/a/b  +  c/d                 →  http://server/a/c/d
http://server/a/b/ +  c/d                 →  http://server/a/b/c/d
http://server/a/b  +  ../c                →  http://server/c
http://server/a/b/ +  ../c                →  http://server/a/c
```

多分最も分かりにくい問題の１つは、ベース URI がスラッシュで終わるかどうかです。

もしベース URI がスラッシュで終わらないなら、相対的な URL が（HTML にマッチする）ベースの親に相対的であると考えられます。

もしベースの URI がスラッシュで終わるなら、相対的な URI はただベースに付加することができます。実際には、階層的な URL の中に組織化されたシステムは常に後続スラッシュをつけてベース URI を指定するべきです。後続スラッシュが付く場合も付かない場合も検索は、結果としてのドキュメントがルートオブジェクトの href の中で暗黙の後続スラッシュがいつも追加されていると考えるべきです。

5.4 Fragment URIs

oBIX ドキュメントに内部のオブジェクトを参照することは一般的にあることです。

これは“#”で始まるフラグメント URI のリファレンスを使って達成されます。

例を考えましょう：

```
<obj href="http://server/whatever/">
  <enum name="switch1" range="#onOff" val="on"/>
```

```
  <enum name="switch2" range="#onOff" val="off"/>
  <list is="obix:Range" href="onOff">
    <obj name="on"/>
    <obj name="off"/>
  </list>
</obj>
```

この例にはフラグメント URI を参照する range ファセットをもつ２つのオブジェクトがあります。“#”で始まっているどんな URI 参照も同じ oBIX ドキュメントの中でオブジェクトを参照すると想定しなくてはなりません。

クライアントがオブジェクトを参照外しするようなもう１つの URI 検索を実行するべきではありません。この場合参照されているオブジェクトは href 属性によって識別されます。

上記の例で”onOff”の href を持っているオブジェクトは両方とも、フラグメント URI のターゲットですが、同じく絶対的なＵＲＩ“ http://server/whatever/onOff ”を持っています。

しかし、おそらく我々はターゲットとなるドキュメント内のフラグメント URI のオブジェクトを持っており、絶対 URI を使用して直接にアドレス指定することはできないでしょう。このような場合 href 属性はそれ自身フラグメント識別子であるべきです。

href 属性が“#”で始まるとき、それが使われることができる唯一の場所が、ドキュメント自身の中であることを意味します:

```
…
  <list is="obix:Range" href="#onOff">
…
```

6 Contracts

コントラクトは、oBIX データソースをモデリングする際に継承パターンを利用する仕組みです。コントラクトとは何であるか？　基本的には、それはちょうど標準 oBIX のオブジェクトです。その他のオブジェクトがテンプレートオブジェクトとして is 属性を使って参照することで、コントラクトオブジェクトを特別にします。

oBIX はコントラクトを何のために使うのか？コントラクトは oBIX における多くの問題を解決します:

- **セマンティクス**: コントラクトは、oBIX 内部で型を定義するのに使用されます。このことにより、ベンダー実装を通じて一貫したセマンティクスを提供するという、共通オブジェクト定義を集合的に同意させます。例えば、Alarm コントラクトは、クライアントソフトウェアが、まったく同じオブジェクト構造を使うことで、どんなベンダーからも正規化されたアラーム情報を引き出すことを保証します。
- **デフォルト**: コントラクトはまた、デフォルト値を指定するための便利な仕組みを用意します。例えば、Alarm コントラクトは、読み込みのたびにネットワーク経由で渡される必要のないすべてのデフォルト値のための仕様を用意します。
- **タイプエクスポート**:Java や C#のような静的型定義言語を使ってシステムを構築することは、多くのベンダーにあることです。コントラクトは、全ての oBIX のクライアントが使用することのできるフォーマットでタイプ情報をエクスポートするための仕組みを用意しています。

なぜ、他のアプローチでなくコントラクトを使うのか？　確かに上記の問題を解決するには多くの方法があります。コントラクトデザインの利点は、柔軟性と単純さにあります。

概念的にコントラクトは、1 個の抽象化で、多くの異なった問題を解決するために、エレガントなモデルを用意します。仕様から見た場合、oBIX XML 構文それ自身を使って新しい抽象モデルを定義できます。実装から見た場合、コントラクトは、クライアントがすでにどのように扱い、構文解析すればよいか知っている、読み込み可能なフォーマットの仕組みを与えます。オブジェクト指向を使い、まったく同じ構文がクラスとインスタンスの両方を表すために使用されます。

6.1 Contract Terminology

コントラクトを議論するために、いくつかの用語を定義することは有益です。

- Contract:は、標準 oBIX XML ドキュメントとして表現される再利用可能なオブジェクト定義である。コントラクトは、oBIX タイプシステムの基盤として使われるテンプレートあるいはプロトタイプです。
- Contract List:はコントラクトオブジェクトに対する 1 個以上の URI のリストです。それは、is, of, in, out 属性の値として使用されます。URI のリストは、空白文字で分離されます。タイプ宣言として、コントラクトリストを考えることができるでしょう。
- Implements:オブジェクトがコントラクトリスト上でコントラクトを指定する場合、オブジェクトは、コントラクトを Implement（実装）すると言います。これは、オブジェクトは、特定のコントラクトの構造とセマンティクスを継承していることを意味します。

908 • Implementation:コントラクトを実装したオブジェクトは、コントラクトの実装と
909 呼ばれます。

910 ## 6.2 Contract List

911 コントラクトリスト属性の構文は、他の oBIX オブジェクトに対する参照 URI のリストと
912 なります。それは、is, of, in, out 属性の値として使用されます。リストの中の URI は、空
913 白文字（ユニコードの 0x20）で分離されています。href 属性のように、コントラクト URI
914 は、絶対 URI、サーバ相対、または同等のフラグメント参照の可能性があります。コント
915 ラクトリストの中の URI は、XML 名前空間接頭辞（プレフィックス）で名前空間を指定
916 されるかもしれません。（7.6 章参照）

917 ## 6.3 Is Attribute

918 オブジェクトは、is 属性を使って、コントラクトを実装します。is 属性の値は、コントラ
919 クトリストです。もし、is 属性が指定されないならば、暗黙のコントラクトリストを決め
920 るために以下のルールが適用されます:

921 • オブジェクトが、list や feed の内部の項目である場合、of 属性で指定されたコン
922 トラクトリストが使われます。

923 • オブジェクトが（名前によって）オブジェクトのコントラクトのひとつとして指
924 定されたオブジェクトをオーバライドした場合、オーバライドされたオブジェク
925 トのコントラクトリストが使われます。

926 • すべての上記のルールが適用できない場合、比較的プリミティブなコントラクト
927 が使用されます。例えば、obj　要素は、暗黙のコントラクト obix:obj をもってお
928 り、real は、暗黙のコントラクト obj:real を持っています。

929

930 bool, int, または str のような要素名は、暗黙のコントラクトに対する糖衣構文（syntactic
931 sugar）です。オブジェクトが、基本要素のひとつを実装した場合、正しい XML 要素名を
932 使用しなくてはなりません。例えば、オブジェクトが obix:int を実装した場合、<obj
933 is="obix:int"/>よりは<int/>として表現されなくてはなりません。それゆえ、obix:bool と
934 obix:int の双方を実装するような、多重の値についてのタイプの実装は不正です。

935 ## 6.4 Contract Inheritance

936 コントラクトは、継承の仕組みであり、それらは古典的な ” それはである ” 関係を確立し
937 ます。理論的には、コントラクトは、タイプの継承を許します。暗黙と明示コントラクト
938 は次のように識別できます。:

939 • **明示コントラクト**：すべての実装が適合するオブジェクト構造を定義します。

940 • **暗黙コントラクト**：コントラクトを伴うセマンティックを定義します。通常、暗
941 黙のコントラクトは、自然言語文を使ってドキュメント化されます。それは、数
942 学的ではないが、人間の解釈に従います。

943 例えば、オブジェクトが Alarm コントラクトを実装するとき、オブジェクトが timestamp
944 オブジェクトを持つことはすぐにわかります。この構造は Alarm の明示コントラクトにあ
945 り、正式に XML で定義されています。しかし Alarm オブジェクトであることが何を意味
946 しているかに対して意味を追加します：そのオブジェクトがアラームイベントについての
947 情報を付加しています。

948 これらのファジー（あいまい）な概念は、機械語では捉えられず、むしろ文として捉える
949 ことができます。

950
951 オブジェクトそれ自体がコントラクトを実装するとき、それは明示的なコントラクトと暗
952 黙のコントラクトに応じなければなりません。それが本当にアラームイベントを意味しな
953 い限り、オブジェクトは obix：Alarm をそのコントラクトリストに入れてはいけません。
954 人間の脳で考えることが勧められること以外に、暗黙のコントラクトについていうことは
955 ありません。それでは、明示的なコントラクトを決定している規則を見ましょう。
956
957 コントラクトの名前のついた子オブジェクトは、実装に自動的に適用されます。実装は、
958 そのコントラクトの子オブジェクトの各々をオーバライドするか、デフォルトを選ぶかも
959 しれません。実装が子オブジェクトを省略するならば、それはコントラクトの値がデフォ
960 ルトになるとされます。実装が（名前による）子オブジェクトを宣言するならば、それは
961 オーバライドされます、そして、実装の値が使われなければなりません。
962 例を見てみましょう:

```
963 <obj href="/def/television">
964   <bool name="power"   val="false"/>
965   <int  name="channel" val="2" min="2" max="200"/>
966 </obj>
967
968 <obj href="/livingRoom/tv" is="/def/television">
969   <int name="channel" val="8"/>
970   <int name="volume"  val="22"/>
971 </obj>
```

972 この例では、URI “ /def/television ” で識別されるコントラクトオブジェクトがあります。
973 電源（power）とチャネル（channel）を格納する 2 個の子オブジェクトがあります。is 属
974 性を介しコントラクトリストで “ /def/television ” を含む居間のテレビインスタンスを指定
975 しています。このオブジェクトではチャネルはデフォルト 2 から 8 にオーバライドされて
976 います。しかし、電源は除外され、デフォルト false を示しています。
977
978 オーバライドは常に name 属性経由で、そのコントラクトと一致します。上記の例で、
979 channel をオーバライドしたのを知っています、なぜなら、オブジェクトを channel とい
980 う名前で宣言したからです。Volume という名前のオブジェクトも宣言されました。
981 Volume は、コントラクトで宣言されていないので、このオブジェクトに対して特定の新
982 しい定義であると仮定します。
983
984 コントラクトの channel オブジェクトは、min と max というファセットを宣言しています。
985 これら 2 個のファセットは、実装によって継承されます。ほとんどすべての属性は、
986 facets, val, of, in, out を含むコントラクトから継承されます。
987 href 属性は、決して継承されません。Null 属性は以下のように継承されます:

988 1. もし null が指定されたら、その明示的な値は使用されます;
989 2. もし val 属性が指定され、null が指定されないと、null は false であると暗示され
990 ます;
991 3. もし、val 属性や null 属性のいずれも指定されないならば、null 属性はコントラク
992 トから継承されます;

993 このことは、null 属性指定を伴わない null でないオブジェクトに null オブジェクトオーバ
994 ライドを暗黙のうちに許します。

6.5 Override Rules

コントラクトのオーバライドは、暗黙と明示コントラクトに従うために必要とされます。暗黙コントラクトは、オブジェクトの実装が、実装したコントラクトとして同じセマンティクスを提供することを意味します。上記の例では、画像の明るさを保持するために channel をオーバライドすることは正しくありません。これは、セマンティクスコントラクトを破壊します。

明示コントラクトをオーバライドすることは、value, facets, または contract list　をオーバライドすることです。しかし、互換性のない型でオブジェクトをオーバライドすることはできません。

例えば、コントラクトが子要素として real を指定した場合、すべての実装は、子要素に対して real を想定する必要があります。特別な場合として、obj は、他のすべての要素タイプに変換可能です。

コントラクトが定義した制限を破ることのないように属性をオーバライドする場合は気を付けなくてはなりません。技術的には、このことは、コントラクトを指定したり、（定義を）狭めることを意味し、コントラクトを一般化したり拡張することを意味しません。この概念は covariance（共変性）と呼ばれます。

例を見ることにしましょう:

```
<int name="channel" val="2" min="2" max="200"/>
```

この例では、コントラクトは、値域２から 200 で宣言されています。このコントラクトのいかなる実装もこの制限を守らなくてはなりません。例えば、min を-100 にオーバライドすることは、値域を広げることなのでエラーとなります。しかし、min を２より大きく、max を 200 より小さくオーバライドするように値域を狭めることはできます。各 facet に対する特定のオーバライド規則は、4.16 章で文書化されています。

6.6 Multiple Inheritance

オブジェクトのコントラクトリストは、実装する多重コントラクト URI を指定できます。これは、現実的にとても一般的であり、多くの場合は必要とされています。

多重コントラクトの実装について２つのトピックがあります:

- **平坦化**: 指定された場合、コントラクトリストは常に平坦化されなくてはなりません。コントラクトが自身のコントラクトリストを持っている場合、役立ちます（セクション 6.6.1 参照）。
- **Mixins:** mixin の設計は、複数のコントラクトがどのように集約されるかについての正しい規則を規定します。この章では、複数のコントラクトが同じ名前を持つ子要素を含んだ場合に矛盾をいかに扱うかを規定しますセクション 6.6.2 参照）。

6.6.1 Flattening

オブジェクト指向言語で継承階層を繋げるようにコントラクトオブジェクト自身がコントラクトを実装することは通常のことです。しかしながらネットワーク経由で oBIX ドキュメントにアクセスすることから、複雑なコントラクト階層について学習を要求される往復のネットワーク要求を最小にしたいと考えます。

この例を考えて見ましょう。

```
<obj href="/A" />
<obj href="/B" is="/A" />
<obj href="/C" is="/B" />
<obj href="/D" is="/C" />
```

この例では、オブジェクト D を最初に読んだ場合、さらにどのコントラクトが実装されているか完全に知るために 3 回の要求が実行されます（C,B,A で一回）。さらには B を実装したオブジェクトをクライアントが探す場合 D をみて判断するのは困難だということです。

これらの問題により、is, of, in, または out 属性が指定された場合　サーバはそれらのコントラクト継承階層をリスト中に平坦化表記することが要求されます。上記例では、正しい表現は以下のようになります:

```
<obj href="/A" />
<obj href="/B" is="/A" />
<obj href="/C" is="/B /A" />
<obj href="/D" is="/C /B /A" />
```

これはクライアントに、D は、これ以上のネットワーク要求を実施せずに C,B,A を実装していることがわかるので、コントラクトリストの迅速なスキャンが可能となります。

6.6.2 Mixins

平坦化は、コントラクトリストが多重コントラクト URI を含んでいるだけでなく、oBIX は mixin メタファーを使って多重継承の伝統的記述をサポートしています。

以下の例を考えましょう:

```
<obj href="acme:Device">
  <str name="serialNo"/>
</obj>

<obj href="acme:Clock" is="acme:Device">
  <op name="snooze">
  <int name="volume" val="0"/>
</obj>

<obj href="acme:Radio" is="acme:Device ">
  <real name="station" min="87.0" max="107.5">
  <int name="volume" val="5"/>
</obj>

<obj href="acme:ClockRadio" is="acme:Radio acme:Clock acme:Device"/>
```

この例では、ClockRadio は、Clock と Radio の双方を実装しています。Clock と Radio の平坦化によって、ClockRadio は Device を実装しています。oBIX では、これは、mixin と呼ばれ、Clock、Radio そして Device が ClockRadio に継承されています。その結果、ClockRadio は 4 つの子要素を継承しています。Serial,snooze,volume と station です。

Mixins は、Java/C#インターフェースと同類の多重継承の形です（oBIX はタイプ継承であり、継承実装でないことを思い出してください）。

Clock と Radio 双方が Device を実装しており、古典的ダイヤモンド継承パターンとなっていることに注意してください。ClockRadio は Device から子要素名 serialNo を継承します。さらには、Clock と Radio 双方は子要素 volume を宣言しています。名前の衝突は、ClockRadio で serialNo と volume が何を意味するかという混乱を引き起こします。

oBIX では、この問題を以下の規則を使用し、コントラクト子要素の平坦化で解決します:

1. コントラクト定義を並んだ順序で処理する。
2. もし子要素が発見されたら、それはオブジェクト定義に組み込まれる。

3. もし以前のコントラクト定義ですでに処理された子要素が発見されたら、以前の定義を優位とします。しかしながら、重複した子要素が前の定義とのコントラクト互換性を欠いていたら、エラーとなります（6.7 章参照）。

上記の例では、Radio.volume は、ClockRadio.volume のために使用した定義ということです。というのは、Radio は Clock より優位（Radio はコントラクトリストで最初に出てくる）だからです。それゆえ、ClockRadio.volume はデフォルト値 5 を持ちます。しかしながら、もし Clock.volume が str として宣言されていれば無効となります、というのは、Radio の定義 int とコントラクト互換でなくなるからです。この場合、ClockRadio は Clock と Radio 双方を実装できません。コントラクトで非互換な名前の衝突を生成しないようにすることは、サーバベンダーの責任です。

リストにおける最初のコントラクトは、特定の意味が与えられますが、それはその定義が他のすべてに優先されるためです。oBIX では、このコントラクトは、プライマリコントラクトと呼ばれます。プライマリコントラクトはコントラクトリストで指定された他のすべてのコントラクトを実装することを勧めます。（これはしばしば、多くのプログラム言語で起こり得ます）これは、クライアントがもし必要なら強力なタイプクラスにオブジェクトを結合させることを簡単にします。明らかに、この推奨はコントラクトオブジェクト自身では意味を持ちません。－コントラクトは自身を実装してはならないからです。

6.7 Contract Compatibility

もう一つのコントラクトリストと共変性的代用可能であるコントラクトリストは、互換性を持つコントラクトであると言われています。リストと演算（操作）のために mixin 規則とオーバライドについて話すとき、コントラクト互換性は役に立つ用語です。それは、前に定義されたオーバライドに類似した、かなり一般的な考え方です‐しかし、個々の facet 属性に適用される規則の代わりに、我々はそれを全コントラクトリストに適用します。もしコントラクトリスト X がコントラクトリスト Y によって定義された値域を狭める場合に限り、X は Y と互換性を持ちます。これは、X が Y をインプリメントするオブジェクトの集合を狭くすることができるが、集合を決して拡大しないことを意味します。コントラクト互換性は、交換可能でありません（X が Y と互換性を持つことが、Y が X と互換性を持つことを意味しているわけではありません）。その定義が大げさであると聞こえるのであれば、次の実際的な規則にまとめることができます：X は新しい URI を Y のリストに加えることができます、しかし、決して何も取り除きません。

6.8 Lists (and Feeds)

list や feed コントラクトから派生する実装は、of 属性を継承します。　他の属性のように、of 属性をオーバライドできます、しかしそれはコントラクト互換の場合に限られます。－コントラクトの of 属性におけるすべての URI を含まなくてはならないが、付加属性を追加することはできます。（第 6.7 節参照）

list と feed にも、暗黙のうちにそれらの内容のコントラクトリストを定義する特別な機能があります。以下の例で、各子要素は、現実に is 属性を各リスト項目上で指定することなしに、def/MissingPerson のコントラクトリストを持ちます。

```
<list of="/def/MissingPerson">
  <obj> <str name="fullName" val="Jack Shephard"/> </obj>
  <obj> <str name="fullName" val="John Locke"/> </obj>
  <obj> <str name="fullName" val="Kate Austen"/> </obj>
</list>
```

1133 list または feed における要素が自身の is 属性を特定するなら、それは of 属性と互換である
1134 コントラクトでなくてはなりません。

7 XML

第４章は oBIX 情報がどのようにモデル化されるかを標準化するのに使用される抽象的なオブジェクト・モデルを規定します。 本章はオブジェクト・モデルが XML でどう表されるかを規定します。

7.1 Design Philosophy

XML 構文の開発にはいろいろなアプローチがありますので，oBIX の XML 構文がどのように設計されたのかという経緯について少し知っておくべきでしょう。M2M の歴史的には、標準的でない拡張は、どんなに良くても価値がなく、一般に不透明です。oBIX の設計原理の１つは、すべてのデータとサービスにとって平等な活躍の場があるように、縦に連なった複数ドメインとベンダーによる拡張を受け入れることです。

この目標を達成するために、obix の XML 構文は、すべて oBIX ドキュメントのために、小さくて、固定されたスキーマをサポートするように設計されています。クライアントエージェントがこの非常に簡単な構文を理解しているなら、それらのオブジェクトが標準の、または、標準的でないコントラクトを実装するかどうかにかかわらずサーバのオブジェクト木へのアクセスは保証されます。

高水準セマンティクスは、コントラクトにより捕らえられます。 コントラクトは、型でオブジェクトに”タグ付け”をして、ダイナミックに適用できます。 これは現場で動的に設定されるシステムのモデリングに非常に役立ちます。 重要なことはコントラクトがオプションとしてクライアントに解釈されるということです。コントラクトは、XML 構文に影響を与えないし、オブジェクト木への基本的アクセスのために、クライアントはコントラクトを使う必要はありません。コントラクトは、オブジェクト木の上の明瞭な単なる抽象化レイヤで、定義された XML 構文です。

7.2 XML Syntax

oBIX XML 構文は抽象的なオブジェクトモデルに非常に密接にマップされます。構文は、以下のようにまとめられます:

- あらゆる oBIX オブジェクトが１つの XML 要素にマップされます;
- 子オブジェクトは子 XML 要素としてマップされます;
- XML 要素名は、ビルドインオブジェクトタイプにマップします;
- オブジェクトに関する他の何もかもが XML 属性として表されます;

4 章のオブジェクトモデル図は、有効な XML 要素とそれぞれの属性を示しています。val オブジェクトが単に val 属性をサポートするオブジェクトのための抽象的な基本型でであって，val 要素ではないことに注意してください。

7.3 XML Encoding

以下の規則は oBIX ドキュメントを XML 化するのに適用されます:

- oBIX ドキュメントは整形式の (well-formed) XML でなくてはなりません;
- oBIX ドキュメントは、最初に XML 宣言でエンコード化方法を指定しなければなりません;

- バイト･オーダー･マークのない UTF-8 符号化を使用することが強く推奨されます;
- oBIX ドキュメントはドキュメントタイプ宣言を含んではいけません。oBIX ドキュメントは内部の、または、外部の部分集合を含むことはできません;
- oBIX ドキュメントは XML 名前空間定義を含まなくてはなりません。慣例は http://obix.org/ns/schema/1.0 をドキュメントのデフォルト名前空間として宣言することです。oBIX が別のタイプの XML ドキュメントの中に埋め込まれるなら、慣例では名前空間接頭語として”o”を使用することになっています。接頭語"obix"を使用してはならないことに注意します(7.6 を見てください);

7.4 XML Decoding

以下の規則は oBIX ドキュメントをデコードするのに適用されます:

- XML1.1 によって定義されるように XML 処理規則に従わなければなりません;
- 整形式の XML でないドキュメントを拒絶しなければなりません;
- パーサはドキュメントタイプ宣言を理解する必要はありません;
- XML 名前空間に関わらず、どんな未知の要素も無視されなくてはなりません;
- XML 名前空間に関わらず、どんな未知の属性も無視されなくてはなりません;

基本的な経験則は以下の通りです。XML ドキュメントの生成時には厳しく、解釈時には自由に（すべて解釈するのではなく必要に応じて解釈すると云う意味です）。 oBIX パーサは意味が解釈できない要素と属性を無視しなければなりません。しかしながら、oBIX パーサは整形式でない(ミスマッチしているタグなどの)XML ドキュメントを決して受け入れてはなりません。

7.5 XML Namespace

oBIX 規格の XML 名前空間は以下のパターンに従うべきです:

```
http://obix.org/ns/{spec}/{version}
```

oBIX バージョン 1.0 のための XML 名前空間は以下の通りです。

```
http://obix.org/ns/schema/1.0
```

明示的記載のない場合、本書のすべての XML はこの名前空間を持つと仮定されます。

7.6 Namespace Prefixes in Contract Lists

oBIX ドキュメントの中に定義された XML 名前空間接頭語は、コントラクトリストの URI の接頭辞として使用されるかもしれません。 コントラクトリストの URI が、”:”コロン文字を従えた定義された XML 接頭辞の文字列で始まるならば、URI は名前空間値を持つ接頭辞で置き換えられることで正規化されます。

"obix"の XML 名前空間接頭語は事前に定義されます。この接頭語は、すべての oBIX で定義されたコントラクトで使用されています。 "obix" 接頭語は文字通り "http://obix.org/def/" に翻訳されます。例えば、URI "obix: bool" は "http://obix.org/def/bool" に翻訳されます。ドキュメントは事前に定義された"obix" 接頭語を使用することで XML 名前空間を定義してはいけません。定義されるなら、それは”http://obix.org/def/”の事前に定義された値によって取って代わられます。 すべての oBIX の定義されたコントラクトは HTTP バインド時には、HTTP URI を使ってアクセスできます。

1213 XML 名前空間接頭語をもつ oBIX ドキュメントが正規化される例:

```
1214 <obj xmlns:acme="http://acme.com/def/" is="acme:CustomPoint obix:Point"/>
1215
1216 <obj is="http://acme.com/def/CustomPoint http://obix.org/def/Point"/>
```

8 Operations

operation は、oBIX オブジェクトに対して実行する行為です。それらは伝統的なオブジェクト指向言語のメソッドと似ています。通常、operation は連続した状態をもつ変数よりむしろコマンドにマップされます。オブジェクトとその現状値を表す value オブジェクトと違い、op 要素は呼び出すことができる operation の定義を表します。

すべての operation が、パラメータとしてちょうど 1 個のオブジェクトを取り、ちょうどその結果 1 個のオブジェクトを返します。 in 属性と out 属性は入力と出力オブジェクトに対してコントラクトリストを定義します。複数の入力オブジェクトや複数の出力オブジェクトが必要な場合、コントラクトを使い複数のオブジェクトを 1 個のオブジェクトにまとめてください。

例:

```
<op href="/addTwoReals" in="/def/AddIn" out="obix:real"/>

<obj href="/def/AddIn">
  <real name="a"/>
  <real name="b"/>
</obj>
```

オブジェクトはコントラクトの 1 つにより operation 定義をオーバライドできます。 しかしながら、新しい入力または出力コントラクトリストは、コントラクト定義とコントラクト互換（6.7 参照）でなくてはなりません。

operation がパラメータを必要としないとき、in オブジェクトに obix:Nil を指定します。operation が何も返さないとき out オブジェクトは obix:Nil を返します。時々、operation は実装でサポートされないコントラクトを継承します。この場合、status 属性を disable に設定します。

operation は、常に自身の href 属性(親の href 属性でなく）経由で呼び出されます。それゆえ、operation は、クライアントに呼び出させるためには、href 属性を指定する必要があります。この規則の共通な例外は、コントラクト自身の定義です。

9 Object Composition

oBIX との比較のための良い比喩は WWW です。 JavaScript や Flash のように創造的なものを無視するなら、基本的に、WWW は、URL でハイパーリンクされた HTML ドキュメントのウェブ（くもの巣）です。 さらに専門的な言い方をするなら、WWW が<p>や、<table>や、 などの HTML 要素のウェブであると言うことができます。

WWW での HTML ドキュメントは、oBIX でのオブジェクトです。 oBIX の論理的なモデルは URI で結びつけられた oBIX オブジェクトのグローバルウェブです。 これらのいくつかの oBIX オブジェクトは、コントラクトやデバイス記述のような静的ドキュメントです。 他の oBIX オブジェクトはリアルタイムデータかサービスを表します。 しかし、それらは皆、oBIX ウェブを形成するために URI で結びつけられます。

個々のオブジェクトはウェブを定義する 2 つの方法で関連付けられます。 オブジェクトは包含あるいは参照により関連付けられます。

9.1 Containment

どんな oBIX オブジェクトも、0 個以上の子オブジェクトを含むことができます。これは bool や int などのプリミティブであると考えられるかもしれないオブジェクトを含みさえします。 すべてのオブジェクトがオブジェクトコントラクト上に存在しないかもしれない新しいオブジェクトを指定することについて制約がなく自由にできます。 包含は XML 要素を入れ子にすることによって、XML 構文で表されます:

```
<obj href="/a/">
  <list name="b" href="b">
    <obj href="b/c">
  </list>
</obj>
```

この例では、"/a"で識別されるオブジェクトは、"/a/b"を含んでいます。さらに"/a/b/c"を含んでいます。子オブジェクトは、name 属性が指定されるかによって、名前付だったり、名前なしだったりします (5.1 参照) 。"/a/b"は名前付き、"/a/b/c"は名前なしです。通常名前付きの子オブジェクトは、レコード, ストラクチャ,またはクラスタイプのフィールドを表すのに使用されます。 名前なしの子オブジェクトはリストでしばしば使用されます。

9.2 References

WWW の比喩に戻りましょう。 WWW は<p>と のような個々の HTML 要素のウェブですが、実際にネットワークの上で個々の<p>要素は受け渡されません。 むしろ、複数の<p>要素を”塊にして”、HTML ドキュメントに入れ、ドキュメントはそのままネットワーク上で受け渡されます。 ドキュメントを結ぶために<a>アンカー要素を使用することでドキュメントの間のリンクを作成します。 これらのアンカーは URI で外のドキュメントを参照し、プレースホルダとしての役目を果たします。

oBIX の参照は、基本的には HTML のアンカーに似ています。それは URI で別の oBIX オブジェクトに”リンクする”プレースホルダとして機能します。 データの小さな木構造をモデル化するために、包含はよく使われる一方、参照は、大きな木構造やオブジェクトのグラフ構造をモデル化するのに使用されます。事実、oBIX Webでは、インターネット上のすべての oBIX オブジェクトを参照を使ってリンクすることができます。

9.3 Extents

oBIX が問題ドメインに適用されるとき、包含か参照のどちらかを使用することで関係をどのようにモデル化するか決めなければなりません。これらの決定には、あなたのモデルがどのように XML に表されて、どのようにネットワークの上でアクセスされるかに直接的な影響があります。包含関係は、XML エンコードとイベントに関して特別なセマンティクスを持つことになります。事実、oBIX はオブジェクトの extent と呼ばれる包含のための用語を新たに作り出します。オブジェクトの extent はその子オブジェクトの木です。 href を持っているオブジェクトだけが extent を持っています。 href のないオブジェクトは常にそれらの先祖の extent の 1 つ以上を含んでいます。

```
<obj href="/a/">
  <obj name="b" href="b">
    <obj name="c"/>
    <ref name="d" href="/d"/>
  </obj>
  <ref name="e" href="/e"/>
</obj>
```

上の例では、‘a'から‘e'という名前のついた 5 個のオブジェクトがあります。‘a'が href を含んでいるので、それには関連する extent があります。それは、b'と‘c'を包含で 'd'、および‘e'を参照で含みます。 同様に、’b’は href を持つので extent であり、‘c'を包含で‘d'を参照で含むことになります。オブジェクト‘c'は、直接 href を提供しませんが、‘a'と‘b'の両方に存在しています。href を持つオブジェクトは、正確にひとつの extent を持ちますが、しかし複数の extent の中で入れ子にされることができることに注意してください。

9.4 XML

オブジェクトを XML にマーシャリングする場合、extent は XML ドキュメントの内部に書かれる必要があります。ドキュメントの外部から参照される唯一有効なオブジェクトは ref 要素自身です。

オブジェクトがコントラクトを実装する場合、extent はコントラクトで定義され、すべてドキュメント内部に書かれる必要があります（コントラクト自身が ref 要素として子オブジェクトを定義しない限り）。子オブジェクトとして、ref オブジェクトを定義している例は、Lobby.about（10.3）です。

10 Networking

oBIX の中心は、そのオブジェクト・モデルとそれに関連する XML 構文です。 しかしながら、oBIX の主なユースケースは、ネットワークの上の情報とサービスにアクセスすることです。 oBIX アーキテクチャはクライアント/サーバネットワークモデルに基づいています:

- **サーバ**: oBIX データとサービスを含むソフトウェアです。 サーバはネットワークの上でクライアントからの要求に応じます。
- **クライアント**: サーバへネットワークを介して要求を送り、oBIX データやサービスにアクセスするソフトウェアです。

クライアントは、要求を受け付けるためのサーバソケットを必要とするサーバ機能の実装を要求されないということは、oBIX の重要な要素でありますが、ソフトウェアは oBIX クライアントとサーバの両方を実装しても構いません。

10.1 Request / Response

すべてのネットワークアクセスが 3 つの要求/応答タイプに要約されます:

- **Read**: 与えられた URI が示すオブジェクトの現状を oBIX XML ドキュメントとして返します。
- **Write**: URI が示す既存のオブジェクトの状態をアップデートする。 書きこむ状態は oBIX XML ドキュメントとしてネットワークの上を通過する。 新しいアップデートされた状態は oBIX XML ドキュメントで返されます。
- **Invoke**: 与えられた URI によって特定された操作を呼び出します。 入力パラメータと出力結果は oBIX XML ドキュメントとしてネットワークの上を通過します。

これらの 3 つの要求/応答がクライアントとサーバの間でどう実装されるかはプロトコルバインディングと呼ばれます。 oBIX 仕様は 2 つの標準プロトコルバインディンングを定義します: HTTP バインディング(第 17 章を参照)と SOAP バインディング(第 18 章を参照)です。しかしながら、どのプロトコルバインディングも、次で述べる同じ read,write,invoke セマンティクスに従わなくてはなりません。

10.1.1 Read

読出し要求は、オブジェクトの URI を指定し、読出し応答はオブジェクトの現在の状態を oBIX ドキュメントとして返します。応答はオブジェクトの完全な extent を含まなければなりません(9.3 を参照)。サーバは読出しが失敗したことを示すために err オブジェクトを返すかもしれません。最も多いエラーは obix:BadUriErr です（10.2 参照）。

10.1.2 Write

書き込み要求は、既存のオブジェクトの現在状態に上書きするために設計されました。書き込み要求は、既存オブジェクトの URI と更新値が必要です。応答は、オブジェクトの更新状態を返します。書き込みが成功した場合、応答はオブジェクトの完全な extent を含まなくてはなりません(9.3 を参照）。書込みが失敗した場合、サーバは失敗を表すエラーオブジェクトを返します。
サーバは、完全にあるいは部分的に書き込みを無視できるので、クライアントは書込みが成功したかどうかをチェックするために応答を確認する必要があります。サーバは書き込みが失敗したことを示すエラーオブジェクトを返すかもしれません。

クライアントは要求にオブジェクトの完全な extent を含む必要はありません。 要求オブジェクト木で明らかに指定されたオブジェクトは上書きされるか、またはサーバの実際のオブジェクト木の上で、”オーバライド”されるべきです。val 属性だけは、書き込みのために指定されるべきです（name 属性などは指定しない）。ファセットを用意している書き込み操作は予期せぬ振る舞いをします。int または real に unit を指定して書き込む場合、書き込まれる値は、読み出されるときと同じ単位でなくてはなりません。―クライアントは、異なるファセットを用意していないし、サーバに単位の自動変換を期待してはなりません。(事実、unit ファセットは要求に含まれません)。

10.1.3 Invoke

呼び出し要求は、操作の引き金として設計されています。呼び出し要求は op オブジェクトの URI と入力引数オブジェクトを指定します。応答は出力オブジェクトを含みます。応答は出力オブジェクトの完全な extent を含まなくてはなりません。(9.3 を参照) サーバは呼び出しが失敗したことを示すために、エラーオブジェクトを返すかもしれません。

10.2 Errors

リクエストエラーは、クライアントまで err 要素を伴って運ばれます。どんな時でも、oBIX サーバは、要求を受け付け、処理できない場合には、サーバはクライアントに err オブジェクトを返さなくてはなりません。 err を伴って有効な oBIX ドキュメントを返すことは、プロトコル特定エラーハンドリング（HTTP応答コードのような）よりも、実行できそうな場合に使用されなくてはなりません。
アプリケーションレベルのエラー処理とデータ伝送を切り離すようなデザインとすることでバッチ要求の部分的な失敗を許容可能となり、プロトコルバインディングの柔軟性が向上します。

いくつかのコントラクトが一般的なエラーとして事前に定義されています:

- **BadUriErr**: 誤った URI か未知の URI のどちらかを示しています;
- **UnsupportedErr**: サーバ実装(サーバがサポートしないコントラクトに基づき定義された操作などの)でサポートされていない要求を示しています;
- **PermissionErr**: クライアントがオブジェクトか操作にアクセスする必要なセキュリティ許可を欠いているのを示すために、使用されます;

これらのエラーのためのコントラクトです:

```
<err href="obix:BadUriErr"/>
<err href="obix:UnsupportedErr"/>
<err href="obix:PermissionErr"/>
```

上記の定義のひとつが、エラーとして意味を持つ場合、err 要素の is 属性を含めるべきです。display 属性で問題の有益な記述を含ませることが強く奨励されます。

10.3 Lobby

すべての oBIX サーバは、obix:Lobby を実装したオブジェクトを用意しなければなりません。
Lobby オブジェクトは、oBIX サーバのエントリポイントとして機能し、oBIX 仕様で規定されたウェルノウンオブジェクトの URI を列挙します。
理論的には、起動時にオブジェクトを発見するために、クライアントが知ることが必要なすべては、Lobby インスタンスに対する URI です。

ベンダーは、他の URI を選択するのは自由であるが、便宜上この URI は、“http://server/obix”とします。Lobby オブジェクトのコントラクトです:

```
<obj href="obix:Lobby">
  <ref name="about" is="obix:About"/>
  <op  name="batch" in="obix:BatchIn" out="obix:BatchOut"/>
  <ref name="watchService" is="obix:WatchService"/>
</obj>
```

Lobby インスタンスは、ベンダーがデータやサービスを発見させるために使われるベンダー特定オブジェクトを配置することができます。

10.4 About

obix:About オブジェクトは、 oBIX サーバについての情報サマリの標準リストです。
クライアントは、Lobby オブジェクトから About オブジェクトの URI を直接発見することができます。
About オブジェクトのコントラクトは以下の通りです:

```
<obj href="obix:About">

  <str name="obixVersion"/>

  <str name="serverName"/>
  <abstime name="serverTime"/>
  <abstime name="serverBootTime"/>

  <str name="vendorName"/>
  <uri name="vendorUrl"/>

  <str name="productName"/>
  <str name="productVersion"/>
  <uri name="productUrl"/>

</obj>
```

以下の子オブジェクトは、oBIX実装についての情報を提供します:

- **obixVersion**: サーバに実装された oBIX 仕様のバージョンを特定する。この文字列は、ドット文字 (Unicode 0x2E)で分離された10進数のリストでなくてはなりません。

以下の子オブジェクトは、サーバ自身の情報を準備します:

- **serverName**: サーバに対して、短いローカライズされた名称を提供します。
- **serverTime**: サーバの現在のローカル時刻を提供します。
- **serverBootTime**: サーバの開始時刻を提供します。 oBIX サーバソフトウェアの開始時刻であり、マシンのブート時刻ではありません。

以下の子オブジェクトは、サーバソフトウェアベンダー名を提供します。

- **vendorName**: oBIX サーバのソフトウェアを実装したベンダーの会社名
- **vendorUrl**: ベンダーWebサイトの URI

以下の子オブジェクトは、サーバを動作させるソフトウェア製品についての情報を提供します:

- **productName**: oBIX サーバソフトウェアの製品名
- **productUrl**: 製品webサイトの URI
- **productVersion**: 製品バージョン番号文字列。ドットで区切られた１０進数を使います。

10.5 Batch

Lobby オブジェクトは、複数のネットワーク要求を単一の操作とするバッチ操作を定義します。
バッチ操作による複数要求はしばしば個々の連続ネットワーク要求を性能改良します。
概して 1 つの大きい要求がネットワークの上の多くの小さい要求よりいつも優れるでしょう。

バッチ要求は標準の oBIX 操作として実装された Read、Write、Invoke 操作の集合です。
プロトコル・バインディングレイヤーでは、Lobby.batch URI を使用する単一の Invoke 要求で表現されます。 サーバへの要求のバッチ処理は、個々の処理が行われるのと等価且つ (記述された) 順序で処理されなくてはなりません。

バッチ操作は、BatchIn オブジェクトを入力して、BatchOut オブジェクトを出力します。

```
<list href="obix:BatchIn" of="obix:uri"/>

<list href="obix:BatchOut" of="obix:obj"/>
```

BatchIn コントラクトは Read、Write または Invoke コントラクトを使用することで特定されたプロセスに要求のリストを指定します:

```
<uri href="obix:Read"/>

<uri href="obix:Write">
  <obj name="in"/>
</uri>

<uri href="obix:Invoke">
  <obj name="in"/>
</uri>
```

BatchOut コントラクトは応答オブジェクトの規則正しいリストをそれぞれの要求に指定します。 例えば、BatchOut における最初のオブジェクトは BatchIn の最初の要求の結果であるに違いありません。操作の失敗はエラー・オブジェクトを使って表現されます。 Read または Write 要求のために BatchIn を経由して渡される各 uri は、対応する結果オブジェクトを BatchIn から識別文字列として使われるhref属性(非正規化あるいは大文字小文字変換は許される)を伴った BatchOut 上に持たなくてはなりません。

部分的な失敗に対処する方法を決めるのはベンダー次第です。一般に冪等な要求 (その操作を何回繰り返しても、1 回だけ実行した時と同じ結果になるもの) は、部分的失敗をエラーを使って表示し、バッチでの以降の要求の処理を続けるべきです。サーバがエラー発生時に次の要求を処理しないと決めるなら、処理されなかったそれぞれの要求にはエラーを返してください。

簡単な例を見てみましょう。

```
<list is="obix:BatchIn">
  <uri is="obix:Read" val="/someStr"/>
  <uri is="obix:Read" val="/invalidUri"/>
  <uri is="obix:Write" val="/someStr">
    <str name="in" val="new string value"/>
  </uri>
</list>

<list is="obix:BatchOut">
```

```
  <str href="/someStr" val="old string value"/>
  <err href="/invalidUri" is="obix:BadUriErr" display="href not found"/>
  <str href="/someStr" val="new string value">
</list>
```

この例では，バッチ要求は、/someStr と/invalidUri に Read 要求を指定していて、/someStr に Write を要求しています。 Write 要求は、”in”という子オブジェクトを含んでいることに注意してください。

サーバは、ちょうどそれぞれの要求 URI あたり 1 個のオブジェクトを指定することによってバッチ要求に応じます。最初の読み出し要求は /someStr によって特定された現行の状態を示す str オブジェクトを返します。2 番目の読み出し要求が無効の URI を含んでいるので、サーバは部分の失敗を示すエラーオブジェクトを返し、そして次の要求を処理します。3 番目の要求は"someStr"に Write するということです。 サーバは、"someStr"の値をアップデートして新しい値を返します。要求が順番に処理されるので最初の要求が"someStr"の元の値を提供して、3 番目の要求が新しい値を含むことに注意してください。 これはまさに私たちが個別にそれぞれのこれらの要求を処理した結果として予想することです。

11 Core Contract Library

本章では、oBIX 仕様に基づいたブロックの生成に使用される、いくつかの基本オブジェクトを定義します。

11.1 Nil

obix:Nil コントラクトは標準の null オブジェクトを定義します。Nil は一般に入出力操作の属性として、入出力パラメータが存在しないことを表すために使われます。定義は以下のようになります。

```
<obj href="obix:Nil" null="true"/>
```

11.2 Range

obix:Range コントラクトは、bool 型や enum 型の範囲を定義するために使用されます。Range は、レンジ項目と呼ばれる 0 あるいはそれ以上のオブジェクトを含む list オブジェクトです。各項目の name は、enum の文字列値として使用される識別子を指定します。項目 ID は、ローカライズしてはならない、レンジ内では 1 回しか使用できないという制限があります。ユーザインタフェースで使用されるローカライズされた文字列を指定するために、オプションとして displayName 属性を使用することができます。定義は次のようになります。

```
<list href="obix:Range" of="obix:obj"/>
```

例:

```
<list href="/enums/OffSlowFast" is="obix:Range">
  <obj name="off"  displayName="Off"/>
  <obj name="slow" displayName="Slow Speed"/>
  <obj name="fast" displayName="Fast Speed"/>
</list>
```

Range は、”true”や”false”の bool 型をテキストとして定義することができます。

```
<list href="/enums/OnOff" is="obix:Range">
  <obj name="true"  displayName="On"/>
  <obj name="false" displayName="Off"/>
</list >
```

11.3 Weekday

obix:Weekday コントラクトは、曜日を表す標準 enum 型です。

```
<enum href="obix:Weekday" range="#Range">
  <list href="#Range" is="obix:Range">
    <obj name="sunday" />
    <obj name="monday" />
    <obj name="tuesday" />
    <obj name="wednesday" />
    <obj name="thursday" />
    <obj name="friday" />
    <obj name="saturday" />
  </list>
</enum>
```

11.4 Month

obix:Month コントラクトは、月を表す標準 enum 型です:

```
<enum href="obix:Month" range="#Range">
  <list href="#Range" is="obix:Range">
    <obj name="january" />
    <obj name="febuary" />
    <obj name="march" />
    <obj name="april" />
    <obj name="may" />
    <obj name="june" />
    <obj name="july" />
    <obj name="august" />
    <obj name="september" />
    <obj name="october"  />
    <obj name="november" />
    <obj name="december" />
  </list>
</enum>
```

11.5 Units

ソフトウェアで計測単位を表すことは、難しい問題です。oBIX は、数学的に単位を定義するために Unit フレームワークを提供します。あらかじめ定義された単位の大規模データベースも用意されています。

物理量の世界では、全ての単位は量や次元に対応しています。よく知られている次元は、７つの基本量：長さ、質量、時間、温度、電流、物質量、光度、で表すことができます。これら７つの次元は、SI 単位で、キログラム（kg）、メータ（m）、秒（sec）、ケルビン（K）、アンペア（A）、モル（mol）、カンデラ（cd）で表されます。

obix:Dimension は、７つの SI 単位の組み合わせを正負のべき乗を使って物理諸量を定義します。

```
<obj href="obix:Dimension">
  <int name="kg"  val="0"/>
  <int name="m"   val="0"/>
  <int name="sec" val="0"/>
  <int name="K"   val="0"/>
  <int name="A"   val="0"/>
  <int name="mol" val="0"/>
  <int name="cd"  val="0"/>
</obj>
```

Dimension オブジェクトは、kg, m, sec, K, A, mol, または cd の０個以上の組み合わせを含みます。これらの組み合わせの各々は、基本 SI 単位のべき乗に対応します。もし、組み合わせが無ければ、その基本量はデフォルトで０を意味しています。例えば、加速度は m/s^2 で oBIX では次のようになります:

```
<obj is="obix:Dimension">
  <int name="m"   val="1"/>
  <int name="sec" val="-2"/>
</obj>
```

同じ次元の単位は、同じ物理量を測るものと考えます。これは、厳密には正しくありませんが、実用には十分です。これにより同じ次元の単位は、換算することができます。換算は、単位を正規化単位に変換する公式で表現されます。各次元に対する正規化単位は、SI 単位自体の比です。例としてエネルギー（ジュール）の正規化単位は m2・kg・s-2 で表されます。キロジュールは、1000 ジュールであり、１ワットアワーは 3600 ジュールです。多くの単位は数学的に正規化単位に変換され、線形公式を使って他の単位に変換されます:

```
unit = dimension • scale + offset
toNormal = scalar • scale + offset
fromNormal = (scalar - offset) / scale
```

```
toUnit = fromUnit.fromNormal( toUnit.toNormal(scalar) )
```

このモデルに合致しないいくつかの単位があります。対数単位や角度を含むものです。しかし、このモデルは、ほとんどの問題に対して実用的な回答を提供できます。このモデルに合わない単位は、すべての基本量のべき乗を0として使用すべきです。アプリケーションはこれら単位の変換を試みるべきではありません。

obix:Unit コントラクトは、次元と正規化の係数を含めて単位オブジェクトを定義します:

```
<obj href="obix:Unit">
  <str  name="symbol"/>
  <obj  name="dimension" is="obix:Dimension"/>
  <real name="scale" val="1"/>
  <real name="offset" val="0"/>
</obj>
```

unit 要素は、symbol, dimension, scale, offset のサブオブジェクトを含みます。

- **symbol**: symbol は、単位を使うための略語を定義します。例えば “ ° F ” は華氏のシンボルです。symbol は常に指定される必要があります。
- **dimension**: dimension は、計測の次元を７つの基本 SI 単位の比として定義します。もし基本量が無い場合、その dimension は、obix:Dimension コントラクトをデフォルトとします。この場合、組み合わせのすべての基本量のべき乗は0です。
- **scale**: scale は、正規化公式のスケール量を定義します。
- **offset**: offset は、正規化公式のオフセット量を定義します。offset のデフォルト値は0です。

display 属性は、現地環境でのローカライズされた単位に対する正式名称を用意します。もしディスプレィ属性が書かれていない場合、クライアントは symbol サブオブジェクトを表示のために使うべきです。

予め定義された単位キロワットの例です:

```
<obj href="obix:units/kilowatt" display="kilowatt">
  <str name="symbol" val="kW"/>
  <obj name="dimension">
    <int name="m" val="2"/>
    <int name="kg" val="1"/>
    <int name="sec" val="-3"/>
  </obj>
  <real name="scale" val="1000"/>
</obj>
```

単位の自動換算はローカライズの問題として考えられます。詳しくは 17.3 章を参照ください。

12 Watches

oBIX で重要なことはリアルタイムでの情報へのアクセスです。我々はクライアントで情報の変化をいち早く受け取る必要があります。しかしながら、我々はクライアントで Web サーバを実行する事や、固定 IP アドレスを付与することを望みません。この問題を解決するために、 oBIX はクライアントがイベントを受信するモデルとして watch を提供します。watch のライフサイクルは以下の通りです:

- クライアントはサーバの WatchService URI の上に make 操作で新しい watch オブジェクトを作成します。サーバは新しい watch オブジェクトを定義して、新しい watch にアクセスするための URI を提供します。
- クライアントは Watch オブジェクト上で監視に利用するオブジェクトを登録（または登録解除）します。
- クライアントは最新のポーリング以降に発生したイベントを取得するために pollChanges 操作を使用して Watch URI を定期的にポーリングします。
- サーバは２つの条件の下で watch を解放します。ひとつはクライアントが delete 操作を使って明示的に watch を解放した場合、もうひとつはクライアントがポーリングに失敗し、あらかじめ決められた時間（リース時間）経過した場合、サーバは自動的に watch を解放することが可能です。あるいは、クライアントが前もって決定された長さの（リース時間と呼ばれる）時間の後にポーリングに失敗した場合にもサーバは自動的に watch を解放するかも知れません。

watch はクライアントが１つ以上のオブジェクトの現在の状態を得るために、リアルタイムキャッシュを保持します。それらは feed オブジェクトからイベントストリームにアクセスするために使われます。加えて、watch はリースによってサーバ上でクライアントごとの状態を管理する標準機能を提供します。

12.1 WatchService

WatchService オブジェクトは新しい watch を作るための URI を提供します。WatchService URI は Lobby オブジェクトから直接利用することができます。

WatchService コントラクトは以下の通りです:

```
<obj href="obix:WatchService">
  <op name="make" in="obix:Nil" out="obix:Watch"/>
</obj>
```

make 操作は、出力として新しい空の Watch オブジェクトを返します。新たに作成された Watch オブジェクトの href はデータセットの読み込みとポーリングを行うために利用することができます。

12.2 Watch

Watch オブジェクトはクライアントからの定期的ポーリング要求や、イベント通知登録の管理を行うために使用されます。コントラクトは以下の通りです：

```
<obj href="obix:Watch">
  <reltime name="lease" min="PT0S" writable="true"/>
  <op name="add"    in="obix:WatchIn" out="obix:WatchOut"/>
  <op name="remove" in="obix:WatchIn"/>
  <op name="pollChanges" out="obix:WatchOut"/>
  <op name="pollRefresh" out="obix:WatchOut"/>
```

```
  <op name="delete"/>
</obj>

<obj href="obix:WatchIn">
  <list name="hrefs" of="obix:WatchInItem"/>
</obj>

<uri href="obix:WatchInItem">
  <obj name="in"/>
</uri>

<obj href="obix:WatchOut">
  <list name="values" of="obix:obj"/>
</obj>
```

watch 操作の多くが 2 つのコントラクトを使います： obix:WatchIn と obix:WatchOut です。クライアントは WatchIn を通してポーリングリストから追加と削除を行うためのオブジェクトを識別します。このオブジェクトは URI のリストを含みます。一般に、これらの URI はサーバ依存（URI）であるべきです。

サーバは add, pollChanges, pollRefresh 操作に対して WatchOut コントラクト経由で応答します。このオブジェクトはポーリング要求されたオブジェクトのリストを含みます。

各オブジェクトは対応する WatchIn 内のクライアントによって識別される URI と正確に同じ文字列を使用して href URI を明示しなければなりません。

サーバは、クライアントから渡された URI についてはいかなる文字の変換や正規化も許可されません。これにより URI 文字列をクライアントソフトウェア上でサーバ応答と照合させるためのハッシュキーとして使うことができます。

12.2.1 Watch.add

一旦 Watch オブジェクトが作成されれば、クライアントは監視のために add を使用して新しいオブジェクトを追加することができます。この操作は、URI のリストを入力し、参照されたオブジェクトの最新値を出力します。返されたオブジェクトはクライアントによって入力されたものと正確に同じ文字列を利用して href を指定することを要求されます。オブジェクトが処理をすることができない場合、部分的な失敗は、それぞれの href と共に err オブジェクトを返すことにより明示されるべきです。後続の URI は 1 つの無効な URI の失敗によって影響を受けるべきではありません。（たとえ監視リストにすでに既存のオブジェクトがあるとしても）add 操作は入力 URI に含まれた明示的でないオブジェクトを決して返すべきではありません。WatchOut の中のオブジェクトの順序と、WatchIn の中の URI の順序は一致する保証はありません -- クライアントは一致のためにキーとして URI を使用しなければなりません。

WatchIn を通して供給された URI はオプションの in パラメータを含むかもしれないことに注意してください。このパラメータは feed オブジェクトのポーリング要求時のみ使用されます。feed はまた、WatchOut 内のヒストリックイベントのリストを返すことが他のオブジェクトとは異なります。feed の詳細がセクション 12.4 で論じられます。

op の href を watch に加えることは無効です、サーバは err を報告しなければなりません。

既に以前から加えられていた URI を watch に追加しようとした場合、サーバは最新の WatchOut の値を返すべきです。けれども、URI がただ 1 度加えられただけであるかのようにポーリング操作を扱ってください ― ポーリングは 1 回のみオブジェクトを返すべき

1752 です。もし多数回に渡って同様の URI が同じ WatchIn リクエストに対して加えられようと
1753 するならばサーバはオブジェクトを１回のみ返すべきです。
1754
1755 注意：URI 最後のスラッシュの欠如は watch 上で問題を起こすことがあります。
1756 最後のスラッシュなしで watch に URI を加えるクライアントが存在することに注意してく
1757 ださい。
1758 クライアントは watch の際に、この URI をローカルのハッシュキーとして使用するでしょ
1759 う。そのためにサーバはクライアントが指定したとおりに URI を使わなくてはなりません。
1760 しかしながら、もしオブジェクトの extent がその子オブジェクトを含むなら、それは同類
1761 の URI を使うことができないでしょう。このような場合にはクライアントが最後のスラッ
1762 シュなしで watch に URI を加えようと試みるとき、サーバは（通常の読み取りリクエスト
1763 にそれを許可するかもしれないけれども）エラー処理を優先して BadUriErr を返すことが
1764 推奨されます。
1765

1766 ### 12.2.2 Watch.remove

1767 クライアントは remove 操作によって watch リストからオブジェクトを削除することがで
1768 きます。URI のリストへ remove を入力すると Nil オブジェクトが返されます。後続の
1769 pollChanges と pollRefresh が削除された URI を含まないようにしなくてはなりません。
1770 watch リストから全ての URI を削除することは可能です; けれどもこの場合は自動的に
1771 watch を解放してはなりません。この場合には普通のポーリングとリースのルールがまだ
1772 適用されます。WatchInItem.in パラメータを remove 操作のために利用することはできま
1773 せん。

1774 ### 12.2.3 Watch.pollChanges

1775 クライアントは pollChanges 操作を使って周期的にサーバをポーリングするべきです。こ
1776 の操作はポーリングに対し変化のあったオブジェクトのリストを返します。サーバは特定
1777 の watch への最新のポーリングリクエスト以降に変更されたオブジェクトのみを返すべき
1778 です。add の場合と同様に、全てのオブジェクトはクライアントが元の add で受け渡した
1779 正しい文字列を使って href を指定しなくてはなりません。もし、サーバ内の extent のい
1780 ずれか一つでも変化したならオブジェクトの全部の extent がクライアントに返されるべき
1781 です。
1782
1783 無効な URI は（add および pollRefresh の）応答に含まれてはなりません。このルールの
1784 例外は正当なオブジェクトが URI スペースから取り除かれるときです。サーバはオブジェ
1785 クトが削除されたことを、BadUriErr コントラクトを含む err で示すべきです。

1786 ### 12.2.4 Watch.pollRefresh

1787 pollRefresh 操作は watch リストのすべてのオブジェクトを強制的に更新します。サーバ
1788 は、クライアントによって加えられ渡される、正確で同じ文字列表現をもつ href を使って、
1789 各オブジェクトとその全 extent を応答として返さなくてはなりません。ポーリングリスト
1790 の無効な URI は err 要素として応答に内包されるべきです。pollRefresh はすべてのオブジ
1791 ェクトのポーリング状態をリセットしますので、次の pollChanges は pollRefresh 呼び出
1792 し以降に状態が変化したオブジェクトを返すだけになります。
1793

12.2.5 Watch.lease

すべての watch は、lease の子オブジェクトによって指定されたリース時間を持っています。もしクライアントからの watch のリクエストがリース時間内に行われなければ、サーバは watch を破棄することができます。リースタイマは新しいポーリングリクエストごとにリセットされます。リース時間内にクライアントからポーリングが 1 回以上行われるのならば、サーバは watch を維持するべきです。

リースタイマは次のリクエストによってリセットされます:watch URI 自身の読み込み、または add, remove, pollChanges, pollRefresh の各操作の呼出です。

クライアントは、lease オブジェクトに書き込みをすることによって異なったリース時間をリクエストするかもしれません（サーバに対して lease の子オブジェクトに href を割り当てるように要求します）。サーバはリクエストを受理するか、リース時間を指定範囲内に変更するか、あるいはリクエストを無視するかを選択できます。すべての書き込みのリクエストは、新しいリース時間を含む応答を返す必要があります。

サーバは BadUriErr コントラクトを含んだ err オブジェクトの応答によって watch の期限切れを報告するべきです。一般的な原則として、リース時間切れ、あるいはクライアントが明示的に delete を呼び出すまでサーバは watch を受理すべきです。

しかしながら、サーバは必要な場合（電力供給停止など）には watch をキャンセルすることが自由にできます。そして新しい watch の再確立はクライアント側から行う必要があります。

12.2.6 Watch.delete

delete 操作は既存の watch を取り消すために使用することができます。

クライアントが oBIX の規約に従うのであれば、自身の watch を削除するべきです。しかしながら、サーバはリースの期限切れによる明示的な削除を除いて、常に正確に(不要な watch を)削除すべきです。

12.3 Watch Depth

それ自身が子オブジェクトを持つオブジェクトに watch が加えられるときクライアントはどのように受信するオブジェクトの階層の“深さ”を知るでしょうか。

oBIX は watch の階層がオブジェクトの extent と一致する必要があります(9.3 参照)。watch がターゲットオブジェクトに加えられるとき、サーバはクライアントに対してターゲットオブジェクトの extent の中の、どのオブジェクトのどんな変更でも通知しなければなりません。もしその extent に feed オブジェクトを含むなら、それらは watch には含まれません – feed と watch の差異についてはセッション 12.4 で議論されます。これは watch が ref と feed 以外の extent 内でその子オブジェクトを包括していることを意味します。

12.4 Feeds

サーバが feed オブジェクトを使ってイベントストリームを明示することができます。

イベントのインスタンスは feed の of 属性によって型を決められます。

クライアントが watch のために feed の href を add することでイベント通知登録し、オプションとして feed の in 属性によって型指定されたインプットパラメータを渡します。

Watch.add から返されたオブジェクトはヒストリカルイベントのリストです（もし”イベントヒストリなし”が利用可能ならば空のリストです）。

WatchOut はこの時点までに起こったすべてのヒストリカルイベントのリストを含みます。

その地理的な場所が変化するとき、イベントを始動させるオブジェクトのシンプルな例を考えましょう：

```
<obj href="/car/">
  <feed href="moved" of="/def/Coordinate"/>
<obj>

<obj href="/def/Coordinate">
  <real name="lat"/>
  <real name="long"/>
</obj>
```

我々は watch に“/car/moved”を加えることによって、moved イベント送信を定期受信します。WatchOut はこの時点まで起こったすべてのヒストリカルイベントのリストを含みます。もしサーバがイベントのヒストリを維持しないなら、このリストは空でしょう：

```
<obj is="obix:WatchIn">
  <list names="hrefs"/>
    <uri val="/car/moved" />
  </list>
</obj>

<obj is="obix:WatchOut">
  <list names="values">
    <feed href="/car/moved" of="/def/Coordinate/" />  <!-- empty history -->
  </list>
</obj>
```

今我々が監視のために pollChanges を呼び出すと、その都度サーバは我々に最新のポーリング以降貯まったイベントのインスタンスのリストを送ります：

```
<obj is="obix:WatchOut">
  <list names="values">
    <feed href="/car/moved" of="/def/Coordinate">
      <obj>
        <real name="lat"  val="37.645022"/>
        <real name="long" val="-77.575851"/>
      </obj>
      <obj>
        <real name="lat"  val="37.639046"/>
        <real name="long" val="-77.61872"/>
      </obj>
    </feed>
  </list>
</obj>
```

feed の of 属性は list の of 属性とまったく同じよう働くことに注意してください。

子オブジェクトのイベントのインスタンスは明示的に上書きされない限り、of によって定義されたコントラクトを継承すると仮定されます。

もしイベントのインスタンスが of コントラクトを上書きするなら、それはコントラクト互換性がなければなりません。セクション 6.8 のルール定義を参照ください。

イベントヒストリを持つ feed を伴う watch 上の pollRefresh 操作を呼び出した場合 pollRefresh は add 操作と同じように全てのヒストリカルイベントを返すべきです。もし

1890 イベントヒストリが利用可能ではないなら、 pollRefresh が普通の pollChanges のように
1891 振る舞い、最後のポーリング以降に起こったイベントを返すべきです。

13 Points

オートメーションシステムに精通する人ならば誰でも、ポイントという用語（産業界ではタグと呼ばれる場合もある）をすぐに理解できます。多くの異なった定義がありますが、一般にポイントとは、そのままセンサーあるいはアクチュエーター（これらはハードポイントと呼ばれる） を指します。

ポイントの概念はソフトウェアの設定ポイント（ソフトポイントと呼ばれる）のようなコンフィギュレーション変数を指すこともあります。ある種のシステムにおいてポイントはアトミックな値であり、また他のシステムにおいてはポイントは膨大なステータスとコンフィギュレーション情報を保持しています。

oBIX の目的は、ベンダー独自なシステムを oBIX 接続可能とすることを試みた場合に、インピーダンス不適合を強要しないでポイントの標準化された表現を取得することです。この要件を満たすために、 oBIX はポイントに対する低レベルの抽象概念 －単純に、一つの関連付けられたステータス情報を伴うプリミティブ値の型として－ を定義します。Point は基本的にはポイントセマンティクスを示すオブジェクトにタグを付けるためだけに使われる目印コントラクトです。

```
<obj href="obix:Point"/>
```

このコントラクトは（次の）基本量タイプでのみ使用されなければなりません：bool, real, enum, str, abstime, reltime です。ポイントは品質情報を伝えるためにステータス属性を使うべきです。次の表はどのように世間一般の制御システムのセマンティクスを値型にマップするべきか示します：

bool	digital point	<bool is="obix:Point" val="true"/>
real	analog point	<real is="obix:Point" val="22" units="obix:units/celsius"/>
enum	multi-state point	<enum is="obix:Point" val="slow"/>

13.1 Writable Points

異なった制御システムが多種多様なセマンティクスを使ってポイントの書き込みを処理します。特定のプライオリティレベルにおいてポイントに書き込む場合もあります。

ポイントがデフォルト値に戻った後の限定された期間にポイントをオーバライドする場合もあります。oBIX 仕様書はベンダーに特定のモデルを課そうと強いることはしません。

むしろ、 oBIX は特殊事例を処理するために、追加 mixin で拡張され得る標準的な WritablePoint コントラクトを提供します。

WritablePoint は書き込む値を収める WritePointIn 構造体を（引数に）取る操作として write を定義します。コントラクトは次の通りです:

```
<obj href="obix:WritablePoint" is="obix:Point">
  <op name="writePoint" in="obix:WritePointIn" out="obix:Point"/>
</obj>

<obj href="obix:WritePointIn">
  <obj name="value"/>
</obj>
```

1930

1931 これは writePoint に渡される値はそのポイントの型と適合することを意味します。例えば
1932 もし WritablePoint が enum で使われるなら、 writePoint は enum を値として渡さなくて
1933 はなりません。

14 History

ほとんどのオートメーションシステムは、時間を通したポイント値の履歴記録をつくるために定期的なポイントデータ収集の機能をもっています。この機能は、ログ、トレンド、ヒストリなど様々な名称で呼ばれています。

oBIX では、ヒストリはタイムスタンプを持つポイントデータのリストとして定義されます。oBIX のヒストリにより提供される特徴として以下のものがあります。

- **History Object**:ヒストリ自身の正規化表現;
- **History Record**:指定された時刻に収集されたポイントのレコード;
- **History Query**: ポイントに対するヒストリデータの標準問い合わせ手順;
- **History Rollup**:ヒストリデータのロールアップ標準手順;

14.1 History Object

標準的な oBIX ヒストリとして（それ自身を）公開することを意図するようなオブジェクトは、コントラクト obix:History を実装します:

```
<obj href="obix:History">
  <int     name="count"  min="0" val="0"/>
  <abstime name="start"  null="true"/>
  <abstime name="end"    null="true"/>
  <op      name="query"  in="obix:HistoryFilter" out="obix:HistoryQueryOut"/>
  <feed    name="feed"   in="obix:HistoryFilter" of="obix:HistoryRecord"/>
  <op      name="rollup" in="obix:HistoryRollupIn" out="obix:HistoryRollupOut"/>
</obj>
```

History の各サブオブジェクトを見ていくことにしましょう:

- **count**:このフィールドは、ヒストリに含まれるレコード数を示します;
- **start**:このフィールドは、最古のレコードのタイムスタンプを示します;
- **end**:このフィールドは、最新のレコードのタイムスタンプを示します;
- **query**:query オブジェクトは、ヒストリレコードを読むためにヒストリに問い合わせるときに使います;
- **feed**:ヒストリレコードのリアルタイム feed にサブスクライブする際に使います;
- **rollup**:このオブジェクトは、ヒストリのロールアップ実行時に使います（数値ヒストリデータに限ります）;

1 時間における 15 分毎の温度データの例は次のとおりです。

```
<obj href="http://x/outsideAirTemp/history/" is="obix:History">
  <int     name="count"  val="5"/>
  <abstime name="start"  val="2005-03-16T14:00"/>
  <abstime name="end"    val="2005-03-16T15:00"/>
  <op      name="query"  href="query"/>
  <op      name="rollup" href="rollup"/>
</obj>
```

14.2 History Queries

各 History オブジェクトは、履歴データを問い合わせるための query 操作を含みます。

14.2.1 HistoryFilter

History.query 入力のコントラクトは以下のとおりです:

```
<obj href="obix:HistoryFilter">
  <int     name="limit"  null=”true”/>
  <abstime name="start"  null="true"/>
  <abstime name="end"    null="true"/>
</obj>
```

以下のフィールドで詳細を記述します:

- **limit**:戻されるべきレコード数の最大値を示す整数です。クライアントはこのフィールドを null でない返されるレコード量のスロットル（絞り弁）として使います。サーバは、指定された制限を越えてレコードを転送してはなりません。しかしながら、サーバは制限以下のデータを返しても良いです。
- **start**:もし null でなければ、このフィールドは、問い合わせ時間範囲の下限（開始）を示します。
- **end**:もし null でなければ、このフィールドは、問い合わせ時間範囲の上限（終了）を示します。

14.2.2 HistoryQueryOut

History.query 出力のコントラクトは以下のとおりです:

```
<obj href="obix:HistoryQueryOut">
  <int     name="count" min="0" val="0"/>
  <abstime name="start" null="true"/>
  <abstime name="end"   null="true"/>
  <list    name="data"  of="obix:HistoryRecord"/>
</obj>
```

History と同様に、各 HistoryQueryOut は count, start, end を返します。しかしヒストリとは異なり、これらの値は問い合わせの結果であり、全ヒストリではありません。実際のヒストリデータは、data フィールドの HistoryRecords にあります。oBIX では子要素の順序は保障されないため、data の後に count が来ることもあり得ます。

14.2.3 HistoryRecord

HistoryRecord コントラクトは、ヒストリ問い合わせの結果のレコードを指定します:

```
<obj href="obix:HistoryRecord">
  <abstime name="timestamp" null="true"/>
  <obj     name="value"     null="true"/>
</obj>
```

典型的な値は、obix:Point で使われるポイント型の値でしょう。

14.2.4 History Query Example

“/outsideAirTemp/history”からの問い合わせ例です:

```
<obj href="http://x/outsideAirTemp/history/query" is="obix:HistoryQueryOut">
  <int name="count" val="5">
  <abstime name="start" val="2005-03-16T14:00"/>
  <abstime name="end"   val="2005-03-16T15:00"/>
  <list name="data" of="#RecordDef obix:HistoryRecord">
    <obj> <abstime name="timestamp" val="2005-03-16T14:00"/>
          <real name="value" val="40"/> </obj>
    <obj> <abstime name="timestamp"  val="2005-03-16T14:15"/>
          <real name="value" val="42"/> </obj>
    <obj> <abstime name="timestamp" val="2005-03-16T14:30"/>
          <real name="value" val="43"/> </obj>
    <obj> <abstime name="timestamp" val="2005-03-16T14:45"/>
          <real name="value" val="47"/> </obj>
```

```
    <obj> <abstime name="timestamp" val="2005-03-16T15:00"/>
          <real name="value" val="44"/> </obj>
  </list>
  <obj href="#RecordDef" is="obix:HistoryRecord">
    <real name="value" units="obix:units/fahrenheit"/>
  </obj>
</obj>
```

注意：上記例では、data リストがすべてのレコードについて共通するファセットを定義するために、どのようにドキュメントローカル（#つき名前）コントラクトを使うかを示しています（コントラクトリストをフラットにすべきではありますが）。

14.3 History Rollups

制御システムは、時刻で収集された生のデータとして履歴データを収集します。しかしながら、多くのアプリケーションはロールアップとよぶ方法で要約された履歴データを使用します。ロールアップ操作は、時間間隔を集約するのに使われます。ヒストリロールアップは、RealPoints のリストとして数値情報を持つようなヒストリに対してのみ適用します。BoolPoints のヒストリのような非数値ヒストリに対するロールアップの問い合わせはエラーとすべきです。

14.3.1 HistoryRollupIn

History.rollup 入力コントラクトは HistoryFilter に周期パラメータを追加したものになります:

```
<obj href="obix:HistoryRollupIn" is="obix:HistoryFilter">
  <reltime name="interval"/>
</obj>
```

14.3.2 HistoryRollupOut

History.rollup 出力コントラクトは:

```
<obj href="obix:HistoryRollupOut">
  <int     name="count" min="0" val="0"/>
  <abstime name="start" null="true"/>
  <abstime name="end"   null="true"/>
  <list name="data" of="obix:HistoryRollupRecord"/>
</obj>
```

HistoryRollupOut オブジェクトは、HistoryRecords ではなく HistoryRollupRecords のリストを返す点を除いて、HistoryQueryOut に非常に良く似ています。

注意：HistoryQueryOut とは異なり、HistoryRollupOut の start（で示される開始時刻）は Rollup 計算処理に使われません。この話題は、次の項目で議論されます。

14.3.3 HistoryRollupRecord

ヒストリ　ロールアップは、HistoryRollupRecords のリストを返します:

```
<obj href="obix:HistoryRollupRecord">
  <abstime name="start"/>
  <abstime name="end"  />
  <int  name="count"/>
  <real name="min" />
  <real name="max" />
  <real name="avg" />
  <real name="sum" />
</obj>
```

子要素の定義は次のとおりです:

- **start**:レコードロールアップ期間の排他的開始時刻（Rollup 処理に使われません）;

- **end**:レコードロールアップ期間の非排他的終了時刻（Rollup 処理に使われます）;
- **count**:このロールアップ期間を計算するために使われるレコード数です;
- **min**:この期間の全レコードの最小値です;
- **max**:この期間の全レコードの最大値です;
- **avg**:この期間の全レコードの平均値です;
- **sum**:この期間の全レコードの合計値です;

14.3.4 Rollup Calculation

ロールアップ計算がどのように実行されるか理解する良い方法は例を見ることです。キロワット値を 15 分ごとに読んで 2 時間取得した計量データのヒストリについて考えます:

```
<obj is="obix:HistoryQueryOut">
  <int     name="count" val="9">
  <abstime name="start" val="2005-03-17T12:00"/>
  <abstime name="end"   val="2005-03-17T14:00"/>
  <list name="data" of="#HistoryDef obix:HistoryRecord">
    <obj> <abstime name="timestamp" val="2005-03-17T12:00"/>
          <real name="value" val="80"> </obj>
    <obj> <abstime name="timestamp" val="2005-03-17T12:15"/>
          <real name="value" val="82"></obj>
    <obj> <abstime name="timestamp" val="2005-03-17T12:30"/>
          <real name="value" val="90"> </obj>
    <obj> <abstime name="timestamp" val="2005-03-17T12:45"/>
          <real name="value" val="85"> </obj>
    <obj> <abstime name="timestamp" val="2005-03-17T13:00"/>
          <real name="value" val="81"> </obj>
    <obj> <abstime name="timestamp" val="2005-03-17T13:15"/>
          <real name="value" val="84"> </obj>
    <obj> <abstime name="timestamp" val="2005-03-17T13:30"/>
          <real name="value" val="91"> </obj>
    <obj> <abstime name="timestamp" val="2005-03-17T13:45"/>
          <real name="value" val="83"> </obj>
    <obj> <abstime name="timestamp" val="2005-03-17T14:00"/>
          <real name="value" val="78"> </obj>
  </list>
  <obj href="#HistoryRecord" is="obix:HistoryRecord">
    <real name="value" units="obix:units/kilowatt"/>
  <obj>
</obj>
```

もし、12 時開始 14 時終了を 1 時間周期でロールアップに対して問い合わせる場合、結果は次のようになります:

```
<obj is="obix:HistoryRollupOut obix:HistoryQueryOut">
  <int     name="count" val="2">
  <abstime name="start" val="2005-03-16T12:00"/>
  <abstime name="end"   val="2005-03-16T14:00"/>
  <list name="data" of="obix:HistoryRollupRecord">
    <obj>
      <abstime name="start" val="2005-03-16T12:00"/>
      <abstime name="end"   val="2005-03-16T13:00"/>
      <int  name="count" val="4"    />
      <real name="min"   val="81"   />
      <real name="max"   val="90"   />
      <real name="avg"   val="84.5" />
      <real name="sum"   val="338"  />
    </obj>
    <obj>
      <abstime name="start" val="2005-03-16T13:00"/>
      <abstime name="end"   val="2005-03-16T14:00"/>
      <int  name="count" val="4"   />
      <real name="min"   val="78"  />
      <real name="max"   val="91"  />
```

```
      <real name="avg"   val="84"  />
      <real name="sum"   val="336" />
    </obj>
  </list>
</obj>
```

もし、計算機を使えるならば、あなたが注意すべき最初のことは、80kw の最初の生データは、ロールアップには使われないということです。これは、開始時刻（の値）は常に使われない(exclusive)からです。開始時刻が使われない理由は、隣接の時間範囲へ離散的なサンプルをまとめているからです。２つの異なったロールアップ期間で、ひとつのデータを含むのは正しくないからです。この問題を避けるために、私たちは常に開始時刻データを排除し、終了時刻データを取り込みます。以下の表は、生データがロールアップ期間でどのように取り扱われるかを示しています:

開始時刻(使われない)	終了時刻(使われる)	含まれるレコード
2005-03-16T12:00	2005-03-16T13:00	82 + 90 + 85 + 81 = 338
2005-03-16T13:00	2005-03-16T14:00	84 + 91 + 83 + 78 = 336

14.4 History Feeds

History コントラクトは、ヒストリレコードのリアルタイム送信を定期受信するために feed を指定します。History.feed は、History.query で使われたものと同じ History.Filter 入力コントラクトを使用します。オブジェクトにヒストリフィードを加えるとき、最初の結果は、入力パラメータによってフィルタされたヒストリレコードのリストを含むべきです。引き続く Watch.pollChanges の呼び出しは、HistoryFilter を満たす最後のポーリング以降に収集された新しいヒストリレコードを返すべきです。

15 Alarming

oBIX アラームは、特長として照会、監視、アラーム確認に対する正規化モデルを指定します。oBIX では、アラームはユーザもしくはもうひとつのアプリケーションによる確認を要求する状態を示します。多くの場合、アラームはアラーム状態を解消してくれる誰か（または何か）に確認を要求します。典型的なアラームのライフサイクルは次のとおりです:

1. **Source Monitoring**：サーバ上で *アラームソース*をモニタするアルゴリズムです。アラームソースは、アラームを生成することのできるポイントの href を伴うオブジェクトを指します。アラームソースの例は、（部屋が暑すぎる場合の）センサーポイント、（ディスクフル状態の）ハードウェアの問題、あるいは（ビルのエネルギー管理で、閾値以上の消費を知らせるような）アプリケーションを含みます。
2. **Alarm Generation**：もしサーバ上のアルゴリズムが、アラームソースがアラーム状態に変化したことを検出した場合、*アラームレコード*が生成されます。各アラームは、href を使ってユニークに識別され obix:Alarm コントラクト使って生成されます。時には、アラームの遷移はオフノーマル（正常からはずれること）として参照されます。
3. **To Normal**：多くのアラームソースは、*ステートフル*と言われます。これは、アラームソースがアラーム状態から脱しノーマル状態になることを指しています。ステートフルアラームは obix:StatefulAlarm コントラクトで実装されます。ソースの遷移がノーマルとなった場合、アラームの normalTimestamp を更新します。
4. **Acknowledgement**：しばしば、ユーザあるいはアプリケーションは、アラームの確認を行います。これらのアラームは、obix:AckAlarm コントラクトで実装されます。アラームが確認されると、ackTimestamp と ackUser を更新します。

15.1 Alarm States

アラーム状態は、次の 2 つの変数によって要約されます:

- **In Alarm**：アラームソースが現在アラーム状態かノーマル状態か示します。この変数は alarm 状態ステートに割り当てられます。
- **Acknowledged**：アラームが確認されたか、未確認かを表します。この変数は、unacked 状態ステートに割り当てられます。

これらの状態のいずれかは他に独立で変化します。例えば、アラームソースはアラームが確認される前でも後でもノーマルになることができます。さらには、確認が起こる前にいくつかのアラーム記録を生成しながら、ノーマルとオフノーマルの遷移が数回起こることも稀ではありません。

注意：全てのアラームがステート（状態）を持つわけではありません。

StatefulAlarm コントラクトや AckAlarm コントラクトを実装しないアラームは、完全にステートレスです。このようなアラームは稀にしか表れません。StatefulAlarm を実装するが、AckAlarm を実装しないアラームは、アラームの状態を持ち、確認の状態を持ちません。一方、AckAlarm を実装するが、StatefulAlarm を実装しないアラームは、確認の状態を持ち、アラームの状態を持ちません。

15.1.1 Alarm Source

アラームソースの現在のアラーム状態は、status 属性で表されます。この属性については 4.16.8 を参照してください。アラームソースは常に status 属性によってステータスを報告することが推奨されます。

15.1.2 StatefulAlarm and AckAlarm

Alarm レコードは、アラームイベントのライフサイクルを要約したものとして使われます。もしアラームが StatefulAlarm を実装するなら、アラームは、オフノーマルからノーマルになるまでの変遷を追跡します。もしアラームが AckAlarm を実装するなら、アラームもまた確認状況を要約します。これは、４個のアラーム状態が存在することを考慮しています:

alarm	acked	normalTimestamp	ackTimestamp
true	false	null	null
true	true	null	non-null
false	false	non-null	null
false	true	non-null	non-null

15.2 Alarm Contracts

15.2.1 Alarm

Alarm コントラクトのコアは次のとおりです:

```
<obj href="obix:Alarm">
  <ref name="source"/>
  <abstime name="timestamp"/>
</obj>
```

子オブジェクトは次のとおりです:

- **source**: アラームソースを特定する URI。ソースは、アラームを生成したエンティティをモデル化した oBIX オブジェクトを参照すべきです。
- **timestamp**: アラームソースがノーマルからオフノーマルに変遷した時刻、そしてアラームレコードが生成された時刻を示します。

15.2.2 StatefulAlarm

アラーム状態からノーマル状態に遷移するかもしれないアラームは、StatefulAlarm コントラクトを実装します:

```
<obj href="obix:StatefulAlarm" is="obix:Alarm">
  <abstime name="normalTimestamp" null="true"/>
</obj>
```

子オブジェクトは次のとおりです:

- **normalTimestamp**: もしアラームソースがアラーム状態にあるなら、このフィールドは null です。そうでない場合、このフィールドはノーマル状態へ遷移した時間を示します。

15.2.3 AckAlarm

確認をサポートするアラームは、AckAlarm コントラクトを実装します:

```
<obj href="obix:AckAlarm" is="obix:Alarm">
  <abstime name="ackTimestamp" null="true"/>
  <str name="ackUser" null="true"/>
  <op name="ack" in="obix:AlarmAckIn" out="obix:AlarmAckOut"/>
</obj>

<obj href="obix:AckAlarmIn">
  <str name="ackUser" null="true"/>
</obj>

<obj href="obix:AckAlarmOut">
  <obj name="alarm" is="obix:AckAlarm obix:Alarm"/>
</obj>
```

子オブジェクトは次のとおりです:

- **ackTimestamp**:アラームが未確認の場合、このフィールドは null です。そうでない場合、確認時刻を示します。
- **ackUser**:アラームが未確認の場合、このフィールドは null です。そうでない場合、このフィールドは確認に責任をもつ人を示す文字列がセットされます。

ack 操作は、プログラムでアラームを確認するのに使われます。クライアントは、AlarmAckIn 経由で ackUser の文字列をオプション指定できます。しかし、サーバはセキュリティ条件によって、このフィールドを無視することができます。例えば、高信頼のクライアントは、自身の ackUser を指定することが許されますが、信頼性の低いクライアントはプロトコルバインドの認証証明書をベースに定義された ackUser を持つかもしれません。ack 操作は、更新されたアラームレコード含む AckAlarmOut を返します。

アラームセットの確認を効率よく行うために Lobby.batch 操作を利用してください。

15.2.4 PointAlarms

アラームソースが obix:Point であることは非常に一般的なことです。各 PointAlarm コントラクトは、アラーム状態を引き起した値を報告する通常の方法で用意されています:

```
<obj href="obix:PointAlarm" is="obix:Alarm">
  <obj name="alarmValue"/>
</obj>
```

alarmValue オブジェクトは、１３章の obix:Point 用に定義されたデータ型のいずれかでなければなりません。

15.3 AlarmSubject

oBIX アラームを実装しているサーバは、AlarmSubject コントラクトを実装した 1 個以上のオブジェクトを提供しなくてはなりません。AlarmSubject コントラクトは、クライアントが検出、照会、監視するアラームセットをカテゴリー分類し、グループ化する機能を有しています。例えば、サーバは全てのアラームに対するひとつの AlarmSubject と、プライオリティや時刻に基づいた他の AlarmSubject を用意することができます。AlarmSubject のコントラクトは次のとおりです:

```
<obj href="obix:AlarmSubject">
  <int     name="count"  min="0" val="0"/>
  <op      name="query" in="obix:AlarmFilter" out="obix:AlarmQueryOut"/>
```

```
  <feed    name="feed"  in="obix:AlarmFilter" of="obix:Alarm"/>
</obj>

<obj href="obix:AlarmFilter">
  <int     name="limit"  null=”true”/>
  <abstime name="start"  null="true"/>
  <abstime name="end"    null="true"/>
</obj>

<obj href="obix:AlarmQueryOut">
  <int     name="count" min="0" val="0"/>
  <abstime name="start" null="true"/>
  <abstime name="end"   null="true"/>
  <list    name="data"  of="obix:Alarm"/>
</obj>
```

AlarmSubject は、History と同じデザインパターンに従います。AlarmSubject は、アクティブなアラームの count を指定します；しかしながら、History と違って、タイムスタンプが付いた start、end を用意していません。それは、時刻の境界でフィルタする AlarmFilter を用い、アラームの現在のリストを読出すための query 操作を含みます。AlarmSubject はまた、アラームイベントを定期受信するのに使われる feed オブジェクトも含んでいます。

15.4 Alarm Feed Example

次の例は、AlarmSubject と共に feed オブジェクトがどのように働くかを示しています:

```
<obj is="obix:AlarmSubject" href="/alarms/">
  <int  name="count" val="2"/>
  <op   name="query" href="query"/>
  <feed name="feed"  href="feed" />
</obj>
```

サーバは、AlarmSubject の下に 2 個のオープンアラームが指定されていることを示しています。クライアントが監視のための AlarmSubject の feed を追加します:

```
<obj is="obix:WatchIn">
 <list names="hrefs"/>
  <uri val="/alarms/feed" />
 </list>
</obj>

<obj is="obix:WatchOut">
 <list names="values">
   <feed href="/alarms/feed" of="obix:Alarm">
    <obj href="/alarmdb/528" is="obix:StatefulAlarm obix:PointAlarm obix:Alarm">
      <ref name="source" href="/airHandlers/2/returnTemp"/>
      <abstime name="timestamp" val="2006-05-18T14:20"/>
      <abstime name="normalTimestamp"  null="null"/>
      <real name="alarmValue" val="80.2"/>
    </obj>
    <obj href="/alarmdb/527" is="obix:StatefulAlarm obix:PointAlarm obix:Alarm">
      <ref name="source" href="/doors/frontDoor"/>
      <abstime name="timestamp" val="2006-05-18T14:18"/>
      <abstime name=" normalTimestamp" null="null"/>
      <real name="alarmValue" val="true"/>
    </obj>
  </feed>
 </list>
</obj>
```

Watch オブジェクトは、2 個のオープンアラームであるアラームイベントのヒストリカルリストを返します。最初のアラームは、AirHandler-2 が返した温度が閾値を越えた状態で

あることを示しています。第二のアラームは、フロントドアが戸止めされて開けっ放しであることを、システムが検出したことを示しています。

システムが、フロントドアが閉じられたことを検出する、そして、アラームは正常な状態に遷移することを想定しましょう。次にクライアントは、（付加的変化やここに表れない新たなアラームとともに）feed list に現われる監視アラームをポーリングします。

```
<obj is="obix:WatchOut">
 <list names="values">
   <feed href="/alarms/feed" of="obix:Alarm">>
    <obj href="/alarmdb/527" is="obix:StatefulAlarm obix:PointAlarm obix:Alarm">
      <ref name="source" href="/doors/frontDoor"/>
      <abstime name="timestamp" val="2006-05-18T14:18"/>
      <abstime name=" normalTimestamp" null="2006-05-18T14:45"/>
      <real name="alarmValue" val="true"/>
    </obj>
   </feed>
 </list>
</obj>
```

16 Security

セキュリティは、広範囲にわたるトピックであり、つぎの論点を含みます:

- **Authentication**: ユーザ・クライアントは誰なのかを確かめること;
- **Encryption**: 詮索好きな目から oBIX ドキュメントを保護すること;
- **Permissions**: オブジェクトを読出す、書込む、または操作を呼び出すためのアクセス権を与える前に、ユーザの許可されている範囲をチェックすること;
- **User Management**: ユーザアカウントと許可レベルを管理すること;

oBIX の基本的思想は、上記項目を仕様の範囲外とすることです。ユーザ認証 と 暗号化はプロトコルバインディングの問題であり、権限付与とユーザ管理は、ベンダー実装の問題です。 oBIX を通して公開されたユーザ管理モデルを定義することは可能ですが、本仕様はユーザ管理に対するいかなる標準規定も定義しないものとします。

16.1 Error Handling

oBIX サーバは認証を実行し、読出し、書込み、リクエスト呼出しを処理する前に許可権限をチェックするため、ユーザの資格証明を利用することを期待されます。一般的な規則として、サーバは、リクエストを実行する権限を持たないクライアントを示すために、obix:PermissionErr コントラクトで err を返します。特にセンシティブなアプリケーションでは、サーバは、信頼できないクライアントに特定のオブジェクトが存在することに気づかれないように、代わりに BadUriErr を返すかもしれません。

16.2 Permission based Degradation

サーバは、クライアントに対して利用可能な権限に基づきクライアントにオブジェクト・モデルを提示するように努めるべきです。この振舞いは、権限に基づく機能制限と呼ばれます。 次の規則は、有効な権限に基づく機能制限をまとめています:

1. オブジェクトが読出し不可の場合、利用可能なオブジェクトを通して発見されないようにしてください。
2. サーバは、同じ権限レベルをもつ標準コントラクトをグループ化すべきです。例えば、start を読めて end を読めないといったように、obix:History の start、end を２つの異なったセキュリティレベルにしないでください。
3. クライアントにとってコントラクトの子オブジェクトが読出し不可の場合、オブジェクトの is 属性上のコントラクトを含まないでください。
4. もしオブジェクトが書込み不可の場合、writable 属性が false であることを（明示的にまたは、デフォルトコントラクトを使って）確認してください。
5. もし可視コントラクトから継承された op が呼出せないとき、op 操作を不可とするために null 属性を true に設定してください。

17 HTTP Binding

HTTP バインドは、HTTP への oBIX 要求の単純な REST マッピングを指定します。読出し要求は、単純な HTTP GET です、これはブラウザの中にその URI をタイプすることによって、簡単にオブジェクトを読出すことができることを意味します。HTTP １.１の全仕様は”RFC2616 Hypertext Transfer Protocol”を参照してください。

17.1 Requests

次のテーブルは どのように oBIX のリクエストを HTTP のメソッドにマップするかを要約したものです:

oBIX Request	HTTP Method	Target
Read	GET	Any object with an href
Write	PUT	Any object with an href and writable=true
Invoke	POST	Any op object

HTTP の要求のために使われた URI は読出し、書込み、呼出しを受けているオブジェクトの URI にマップされなければなりません。

読出し要求は簡素な HTTP GET を使用し oBIX ドキュメントを結果として返します。書込みと呼出しは、それぞれ PUT と POST メソッドで実装されます。

入力はサーバに oBIX ドキュメントとして渡され、結果は oBIX ドキュメントとして返されます。

もし oBIX サーバが要求を処理するなら、結果として生じる oBIX ドキュメントを 200 OK ステータスコードと共に返さなくてはなりません。

たとえリクエストが失敗して結果としてサーバが err オブジェクトを返してきたとしても 200 ステータスコードを使わなくてはなりません。

クライアントとサーバの間でやり取りする oBIX ドキュメントは、”text/xml”の MIME タイプを Content-Type HTTP ヘッダーに指定するべきです。

クライアントとサーバは、標準的な XML エンコーディングルールを使ってネットワーク通信される oBIX ドキュメントをコード化しなくてはなりません。byte-order マークなしの UTF8 を使用することが強く推奨されます。指定する場合、Content-Encoding HTTP ヘッダーは XML エンコーディングにマッチさせなければなりません。

17.2 Security

多数の標準規約が HTTP に認証と暗号化サービスを提供するよう設計されています。oBIX HTTP の実装を適用する場合、次のような既存の標準規約を利用してください:

- RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication
- RFC 2818 - HTTP Over TLS (HTTPS)

- RFC 4346/2246 – The TLS Protocol (Transport Layer Security)

17.3 Localization

サーバは、クライアントのエージェントの望ましいロケールに基づいて適切なデータにローカライズするべきです。ローカライゼーションは display と displayName 属性を含みます。クライアントの望ましいロケールは、認証を通して、あるいは Accept-Language HTTP ヘッダーを経由して決定されるでしょう。提案しているアルゴリズムにおいて、認証されたユーザの好みのロケールがサーバのユーザ・データベースで設定されているかどうかをチェックしてください、もし未設定なら Accept-Language ヘッダーから得たロケールで代替してください。

ローカライゼーションは、単位の自動変換を含むかもしれません。

例えば、もし認証されたユーザがメートル法に対して英国式の単位系の設定を好むならば、サーバは、関連づけられた unit ファセットの値を好みの単位系に変換しようとするでしょう。

18 SOAP Binding

SOAP バインディングは SOAP オペレーションを 3 つの oBIX のリクエストタイプのそれぞれにマップします：read、write、invoke です。

HTTP バインドのように読み出しはすべてのオブジェクトでサポートされ、書き込みはそのオブジェクトの writable 属性が true である場合にサポートされます、そして呼び出しはオペレーションに限ってサポートされます。入力と出力の各リクエストはターゲットオブジェクトに固有のものです。

HTTP バインドとは異なり、リクエストはターゲットオブジェクトの URI を通してアクセスするのではなく、その代わりに SOAP エンベロープの body 内にエンコードされたオブジェクトの URI と共に SOAP サーバの URI を介して行われます。

18.1 SOAP Example

以下は oBIX サーバの About オブジェクトへの SOAP リクエストです:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Body>
  <read xmlns="http://obix.org/ns/wsdl/1.0"
        href="http://localhost/obix/about" />
</env:Body>
</env:Envelope>
```

上のリクエストに対する応答例:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Body>
  <obj name="about"
       href="http://localhost/obix/about/"
       xmlns="http://obix.org/ns/schema/1.0">
     <str name="obixVersion" val="1.0"/>
     <str name="serverName" val="obix"/>
     <abstime name="serverTime" val="2006-02-08T09:40:55.000+05:00"/>
     <abstime name="serverBootTime" val="2006-02-08T09:33:31.980+05:00"/>
     <str name="vendorName" val="Acme, Inc."/>
     <uri name="vendorUrl" val="http://www.acme.com"/>
     <str name="productName" val="Acme oBIX Server"/>
     <str name="productVersion" val="1.0.3"/>
     <uri name="productUrl" val="http://www.acme.com/obix"/>
  </obj>
 </env:Body>
</env:Envelope>
```

18.2 Error Handling

oBIX 仕様は SOAP フォールトを定義しません。もしリクエストが oBIX サーバによって処理されるなら、正当な oBIX ドキュメントが err オブジェクトを通して示された失敗と共に返されるべきです。

18.3 Security

WS-I Basic Profile 1.0 内のセキュリティに関する推奨を参照してください:

http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html#security

18.4 Localization

適用可能であるとき、 SOAP バインドは HTTP バインドのために定義されたローカリゼーションパターンに従うべきです(17.3 参照)。

18.5 WSDL

oBIX スキーマは、WSDL ドキュメントのタイプセクションで読み込まれます。

サーバ実装は、(oBIX WSDL の)標準のドキュメントにおいて欠如している schemaLocation 属性を供給することを考慮しても良いでしょう。

サービス要素が標準の oBIX WSDL からは欠けています。この要素はネットワークアドレスと SOAP サーバのインスタンスを紐付けします。各（SOAP サーバ）インスタンスは WSDL ドキュメントのインスタンス独自のサービスセクションを提供しなければならないでしょう。以下は WSDL サービス要素の例です:

```
<wsdl:service name="obix">
  <wsdl:port name="obixPort" binding="tns:obixSoapBinding">
    <soap:address location="http://localhost/obix/soap"/>
  </wsdl:port>
</wsdl:service>
```

標準の oBIX WSDL:

```
<wsdl:definitions targetNamespace="http://obix.org/ns/wsdl/1.0"
                  xmlns="http://obix.org/ns/wsdl/1.0"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:obix="http://obix.org/ns/schema/1.0">
 <wsdl:types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              elementFormDefault="qualified"
              targetNamespace="http://obix.org/ns/wsdl/1.0">
   <xsd:import namespace="http://obix.org/ns/schema/1.0"/>
   <xsd:complexType name="ReadReq">
    <xsd:attribute name="href" type="xsd:anyURI"/>
   </xsd:complexType>
   <xsd:complexType name="WriteReq">
    <xsd:complexContent>
     <xsd:extension base="ReadReq">
      <xsd:sequence>
       <xsd:element ref="obix:obj" maxOccurs="1" minOccurs="1"/>
      </xsd:sequence>
     </xsd:extension>
    </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="InvokeReq">
    <xsd:complexContent>
     <xsd:extension base="ReadReq">
      <xsd:sequence>
       <xsd:element ref="obix:obj" maxOccurs="1" minOccurs="1"/>
      </xsd:sequence>
     </xsd:extension>
    </xsd:complexContent>
   </xsd:complexType>
   <xsd:element name="read" type="ReadReq"/>
   <xsd:element name="write" type="WriteReq"/>
   <xsd:element name="invoke" type="InvokeReq"/>
  </xsd:schema>
 </wsdl:types>
```

```
 <wsdl:message name="readSoapReq">
  <wsdl:part name="body" element="read"/>
 </wsdl:message>
 <wsdl:message name="readSoapRes">
  <wsdl:part name="body" element="obix:obj"/>
 </wsdl:message>
 <wsdl:message name="writeSoapReq">
  <wsdl:part name="body" element="write"/>
 </wsdl:message>
 <wsdl:message name="writeSoapRes">
  <wsdl:part name="body" element="obix:obj"/>
 </wsdl:message>
 <wsdl:message name="invokeSoapReq">
  <wsdl:part name="body" element="invoke"/>
 </wsdl:message>
 <wsdl:message name="invokeSoapRes">
  <wsdl:part name="body" element="obix:obj"/>
 </wsdl:message>
 <wsdl:portType name="oBIXSoapPort">
  <wsdl:operation name="read">
   <wsdl:input message="readSoapReq"/>
   <wsdl:output message="readSoapRes"/>
  </wsdl:operation>
  <wsdl:operation name="write">
   <wsdl:input message="writeSoapReq"/>
   <wsdl:output message="writeSoapRes"/>
  </wsdl:operation>
  <wsdl:operation name="invoke">
   <wsdl:input message="invokeSoapReq"/>
   <wsdl:output message="invokeSoapRes"/>
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="oBIXSoapBinding" type="oBIXSoapPort">
  <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="read">
   <soap:operation soapAction="http://obix.org/ns/wsdl/1.0/read"
                   style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="write">
   <soap:operation soapAction="http://obix.org/ns/wsdl/1.0/write"
                   style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="invoke">
   <soap:operation soapAction="http://obix.org/ns/wsdl/1.0/invoke"
                   style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
</wsdl:definitions>
```

Appendix A. Revision History

Rev	Date	By Whom	What
wd-0.1	14 Jan 03	Brian Frank	Initial version
wd-0.2	22 Jan 03	Brian Frank	
wd-0.3	30 Aug 04	Brian Frank	Move to Oasis, SysService
wd-0.4	2 Sep 04	Brian Frank	Status
wd-0.5	12 Oct 04	Brian Frank	Namespaces, Writes, Poll
wd-0.6	2 Dec 04	Brian Frank	Incorporate schema comments
wd-0.7	17 Mar 05	Brian Frank	URI, REST, Prototypes, History
wd-0.8	19 Dec 05	Brian Frank	Contracts, Ops
wd-0.9	8 Feb 06	Brian Frank	Watches, Alarming, Bindings
wd-0.10	13 Mar 06	Brian Frank	Overview, XML, clarifications
wd-0.11	20 Apr 06	Brian Frank	10.1 sections, ack, min/max
wd-0.11.1	28 Apr 06	Aaron Hanson	WSDL Corrections
wd-0.12	22 May 06	Brian Frank	Status, feeds, no deltas
wd-0.12.1	29 Jun 06	Brian Frank	Schema, stdlib corrections
obix-1.0-cd-02	30 Jun 06	Aaron Hansen	OASIS document format compliance.
obix-1.0-cs-01	18 Oct 06	Brian Frank	Public review comments