7 Contracts

OBIX Contracts are used to define inheritance in OBIX Objects. A Contract is a template, defined as an OBIX Object, that is referenced by other Objects. These templates are referenced using the `is` attribute. Contracts solve several important problems in OBIX:

Semantics	Contracts are used to define "types" within OBIX. This lets us collectively agree on common Object definitions to provide consistent semantics across vendor implementations. For example the `Alarm` Contract ensures that Client software can extract normalized alarm information from any vendor's system using the exact same Object structure.
Defaults	Contracts also provide a convenient mechanism to specify default values. Note that when serializing Object trees to XML (especially over a network), defaults are typically not allowed, in order to keep Client processing simple.
Type Export	OBIX will be used to interact with existing and future control systems based on statically-typed languages such as Java or C#. Contracts provide a standard mechanism to export type information in a format that all OBIX Clients can consume.

Table 7-1. Problems addressed by Contracts.

The benefit of the Contract design is its flexibility and simplicity. Conceptually Contracts provide an elegant model for solving many different problems with one abstraction. One can define new abstractions using the OBIX syntax itself. Contracts also give us a machine readable format that Clients already know how to retrieve and parse –the exact same syntax is used to represent both a class and an instance.

7.1 Contract Terminology

Common terms that are useful for discussing Contracts are defined in the following Table. Contracts are the templates or prototypes used as the foundation of the OBIX type system. They may contain both syntactical and semantic behaviors.

Term	Definition
Contract Definition	A reusable Object definition expressed as a standard OBIX Object.
Contract	A list of one or more URIs to Contract Objects. The list of URIs is separated by the space character. It is used as the value of the `is`, `of`, `in` and `out` attributes.
Contract Element	A single URI in a Contract.
Implements	When an Object includes a Contract Element in a Contract, the Object is said to *implement* the Contract. This means that the Object is inheriting both the structure and semantics of the specified Contract.
Implementation	An Object which implements a Contract or Contract Element is said to be an *implementation* of that Contract.

Table 7-2. Contract terminology.

William Cox 4/29/2015 10:38 PM
Deleted: Contract ... [1]

William Cox 4/29/2015 10:20 PM
Comment [4]: These appear to be the only use of type "contract" – see diagram update.

William Cox 4/22/2015 11:45 PM
Comment [5]: Used only in this section and 7.6.1

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: specifies

William Cox 4/29/2015 10:38 PM
Deleted: its

William Cox 4/29/2015 10:38 PM
Deleted: List

7.2 Contract List

The syntax of a Contract is a list of URI references to other OBIX Objects[1]. The Contract Elements within the Contract MUST be separated by the space character (Unicode 0x20). Just like the `href` attribute, a Contract Element URI MAY be an absolute URI, Server relative, or even a fragment. The URIs within a Contract may be individually scoped with an XML namespace prefix (see “Namespace Prefixes in Contract Lists” in the **[OBIX Encodings]** document).

A Contract is not an `obix:list` type described in Section 4.3.2. It is a string with special structure and semantics regarding the space-separated URIs.

The Contract is used as the value of the `is`, `of`, `in` and `out` attributes. An example of a point that implements multiple Contracts and advertises this through its Contract is:

```
<real val="70.0" name="setpoint" is="obix:Point obix:WritablePoint acme:Setpoint"/>
```

From this example, we can see that this 'setpoint' Object implements the Point and WritablePoint Contracts that are described in this specification (Section 13). It also implements a separate Contract defined with the `acme` namespace called Setpoint. A consumer of this Object can rely on the fact that it has all of the syntactical and semantic behaviors of each of these Contracts, and one can interact with any of these behaviors.

An example of an `obix:list` that uses Contract List in its `of` attribute to describe the type of items contained in the `obix:list` is:

```
<list name="Logged Data" of="obix:Point obix:History">
  <real name="spaceTemp"/>
  <str val="Whiskers on Kittens"/>
  <str val="Bright Copper Kettles"/>
  <str val="Warm Woolen Mittens"/>
</list>
```

7.3 Is Attribute

An Object defines the Contract it implements via the `is` attribute. The value of the `is` attribute is a Contract. If the `is` attribute is unspecified, then the following rules are used to determine the implied Contract Elements:

- If the Object is an item inside a `list` or `feed`, then the Contract Element specified by the `of` attribute is used.
- If the Object overrides (by name) an Object specified in one of its Contract Elements, then the Contract of the overridden Object is used.
- If all the above rules fail, then the respective Contract Element is used. For example, an `obj` element has an implied Contract of `obix:obj` and `real` an implied Contract of `obix:real`.

Element names such as `bool`, `int`, or `str` are abbreviations for implied Contracts. However if an Object implements one of the primitive types, then it MUST use the correct OBIX type name. If an Object implements `obix:int`, then it MUST be expressed as `<int/>`, and MUST NOT use the form `<obj`

[1] This implies that self-referential or loops in references in Contract Elements is forbidden. NEED CONFORMANCE CLAUSE.

William Cox 4/29/2015 10:38 PM
Deleted: List attribute

William Cox 4/29/2015 10:38 PM
Deleted: URIs

William Cox 4/29/2015 10:38 PM
Deleted: list

William Cox 4/29/2015 10:38 PM
Deleted: can

William Cox 4/29/2015 10:38 PM
Deleted: reference

William Cox 4/29/2015 10:16 PM
Comment [6]: Not sure whether this is correct.

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/22/2015 11:41 PM
Comment [7]: Why is the type not Contract List instead of “contract” in the schema?

William Cox 4/29/2015 10:38 PM
Deleted: group of

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: l

William Cox 4/29/2015 10:38 PM
Deleted: Contracts

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: Contracts

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: primitive

is="obix:int"/>. An Object MUST NOT implement multiple value types, such as implementing both obix:bool and obix:int.

7.4 Contract Inheritance

7.4.1 Structure vs Semantics

Contracts are a mechanism of inheritance – they establish the classic "is a" relationship. In the abstract sense a Contract allows inheritance of a *type*. One can further distinguish between the explicit and implicit Contract:

Explicit Contract	Defines an object structure which all implementations must conform with. This can be evaluated quantitatively by examining the Object data structure.
Implicit Contract	Defines semantics associated with the Contract. The implicit Contract is typically documented using natural language prose. It is qualitatively interpreted, rather than quantitatively interpreted.

Table 7-3. Explicit and Implicit Contracts.

For example when an Object implements the Alarm Contract, one can immediately infer that it will have a child called timestamp. This structure is in the explicit contract of Alarm and is formally defined in its encoded definition. But semantics are also attached to what it means to be an Alarm Object: that the Object is providing information about an alarm event. These subjective concepts cannot be captured in machine language; rather they can only be captured in prose.

When an Object declares itself to implement a Contract it MUST meet both the explicit Contract and the implicit Contract. An Object MUST NOT put obix:Alarm in its Contract unless it really represents an alarm event. Interpretation of Implicit Contracts generally requires that a human brain be involved, i.e., they cannot in general be consumed with pure machine-to-machine interaction.

William Cox 4/29/2015 10:38 PM
Deleted: List

7.4.2 Overriding Defaults

A Contract's named children Objects are automatically applied to implementations. An implementation may choose to *override* or *default* each of its Contract's children. If the implementation omits the child, then it is assumed to default to the Contract's value. If the implementation declares the child (by name), then it is overridden and the implementation's value SHOULD be used. Let's look at an example:

```
<obj href="/def/television">
  <bool name="power"   val="false"/>
  <int  name="channel" val="2" min="2" max="200"/>
</obj>

<obj href="/livingRoom/tv" is="/def/television">
  <int name="channel" val="8"/>
  <int name="volume"  val="22"/>
</obj>
```

In this example a Contract Object is identified with the URI "/def/television". It has two children to store power and channel. The living room TV instance includes "/def/television" in its Contract via the is attribute. In this Object, channel is *overridden* to 8 from its default value of 2. However since power was omitted, it is implied to *default* to false.

William Cox 4/29/2015 10:38 PM
Deleted: List

An override is always matched to its Contract via the name attribute. In the example above it was clear that 'channel' was being overridden, because an Object was declared with a name of 'channel'. A second Object was also declared with a name of 'volume'. Since volume wasn't declared in the Contract, it is assumed to be a new definition specific to this Object.

7.4.3 Attributes and Facets

Also note that the Contract's channel Object declares a `min` and `max` Facet. These two Facets are also inherited by the implementation. Almost all attributes are inherited from their Contract including Facets, `val`, `of`, `in`, and `out`. The `href` attribute is never inherited. The `null` attribute inherits as follows:

1. If the `null` attribute is specified, then its explicit value is used;
2. If a `val` attribute is specified and `null` is unspecified, then `null` is implied to be false;
3. If neither a `val` attribute or a `null` attribute is specified, then the `null` attribute is inherited from the Contract;
4. If the `null` attribute is specified and is true, then the `val` attribute is ignored.

This allows us to implicitly override a null Object to non-null without specifying the `null` attribute.

7.5 Override Rules

Contract overrides are REQUIRED to obey the implicit and explicit Contract. Implicit means that the implementation Object provides the same semantics as the Contract it implements. In the example above it would be incorrect to override channel to store picture brightness. That would break the semantic Contract.

Overriding the explicit Contract Element means to override the value, Facets, or Contract. However one can never override the Object to be an incompatible value type. For example if the Contract specifies a child as `real`, then all implementations must use `real` for that child. As a special case, `obj` may be narrowed to any other element type.

William Cox 4/29/2015 10:38 PM
Deleted: List

One must also be careful when overriding attributes to never break restrictions the Contract has defined. Technically this means the value space of a Contract can be *specialized* or *narrowed*, but never *generalized* or *widened*. This concept is called *covariance*. Returning to the example from above:

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

In this example the Contract has declared a value space of 2 to 200. Any implementation of this Contract must meet this restriction. For example it would an error to override `min` to –100 since that would widen the value space. However the value space can be narrowed by overriding `min` to a number greater than 2 or by overriding `max` to a number less than 200. The specific override rules applicable to each Facet are documented in section 4.2.7.

7.6 Multiple Inheritance

An Object's Contract may specify multiple Contract Element URIs which it implements. This is actually quite common - even required in many cases. There are two terms associated with the implementation of multiple Contracts:

Flattening	Contract SHOULD always be *flattened* when specified. This comes into play when a Contract Element has its own Contract (Section 7.6.1).
Mixins	The mixin design specifies the exact rules for how multiple Contracts are merged together. This section also specifies how conflicts are handled when multiple Contracts contain children with the same name (Section 7.6.2).

Table 7-4. Contract inheritance.

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: to implement

William Cox 4/29/2015 10:38 PM
Deleted: Lists

William Cox 4/29/2015 10:38 PM
Deleted: List

7.6.1 Flattening

It is common for Contract Objects themselves to implement Contracts, just like it is common in OO languages to chain the inheritance hierarchy. However due to the nature of accessing OBIX documents over a network, it is often desired to minimize round trip network requests which might be needed to "learn" about a complex Contract hierarchy. Consider this example:

```
<obj href="/A" />
```

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

In this example if an OBIX Client were reading Object D for the first time, it would take three more requests to fully learn what Contracts are implemented (one for C, B, and A). Furthermore, if the Client was just looking for Objects that implemented B, it would difficult to determine this just by looking at D.

Because of these issues, Servers are REQUIRED to flatten their Contract inheritance hierarchy into a list when specifying the `is`, `of`, `in`, or `out` attributes. In the example above, the correct representation would be:

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

This allows Clients to quickly scan D's Contract to see that D implements C, B, and A without further requests.

Because complex Servers often have a complex Contract hierarchy of Object types, the requirement to flatten the Contract hierarchy can lead to a verbose Contract. Often many of these Contracts Elements are from the same namespace. For example:

```
<obj name="VSD1" href="acme:VSD-1" is="acmeObixLibrary:VerySpecificDevice1
acmeObixLibrary:VerySpecificDeviceBase acmeObixLibrary:SpecificDeviceType
acmeObixLibrary:BaseDevice acmeObixLibrary:BaseObject"/>
```

To save space, Servers MAY choose to combine the Contract Elements from the same namespace and present the Contract with the namespace followed by a colon, then a brace-enclosed list of Contract names:

```
<real name="writableReal" is="obix:{Point WritablePoint}"/>

<obj name="vsd1" href="acme:VSD-1" is="acmeObixLibrary:{VerySpecificDevice1
VerySpecificDeviceBase SpecificDeviceType BaseDevice BaseObject}"/>
```

Clients MUST be able to consume this form of the Contract and expand it to the standard form.

7.6.2 Mixins

Flattening is not the only reason a Contract might contain multiple Contract Elements. OBIX also supports the more traditional notion of multiple inheritance using a mixin approach as in the following example:

```
<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"/>
```

In this example `ClockRadio` implements both `Clock` and `Radio`. Via flattening of `Clock` and `Radio`, `ClockRadio` also implements `Device`. In OBIX this is called a *mixin* – `Clock`, `Radio`, and `Device` are mixed into (merged into) `ClockRadio`. Therefore `ClockRadio` inherits four children: `serialNo`, `snooze`, `volume`, and `station`. Mixins are a form of multiple inheritance akin to Java/C# interfaces (remember OBIX is about the type inheritance, not implementation inheritance).

Note that `Clock` and `Radio` both implement `Device`. This inheritance pattern where two types both inherit from a base, and are themselves both inherited by a single type, is called a "diamond" pattern from the shape it takes when the class hierarchy is diagrammed. From `Device`, `ClockRadio` inherits a child named `serialNo`. Furthermore notice that both `Clock` and `Radio` declare a child named `volume`. This

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: Contracts

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:25 PM
Comment [8]: Note that this is additional conformance requirements on the string; not in the schema IMO.

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: URIs

naming collision could potentially create confusion for what `serialNo` and `volume` mean in `ClockRadio`.

OBIX solves this problem by flattening the Contract's children using the following rules:

1. Process the Contract definitions in the order they are listed
2. If a new child is discovered, it is mixed into the Object's definition
3. If a child is discovered that has already been processed via a previous Contract definition, then the previous definition takes precedence. However it is an error if the duplicate child is not *Contract compatible* with the previous definition (see Section 7.7).

In the example above this means that `Radio.volume` is the definition used for `ClockRadio.volume`, because `Radio` has a higher precedence than `Clock` (it is first in the Contract). Thus `ClockRadio.volume` has a default value of "5". However it would be invalid if `Clock.volume` were declared as `str`, since it would not be Contract compatible with `Radio`'s definition as an `int` – in that case `ClockRadio` could not implement both `Clock` and `Radio`. It is the Server vendor's responsibility not to create incompatible name collisions in Contracts.

The first Contract Element in a Contract is given special significance since its definition trumps all others. In OBIX this Contract Element is called the *Primary Contract Element*. For this reason, the Primary Contract Element SHOULD implement all the other Contracts specified in the Contract (this actually happens quite naturally by itself in many programming languages). This makes it easier for Clients to bind the Object into a strongly typed class if desired. Contracts MUST NOT implement themselves nor have circular inheritance dependencies.

7.7 Contract Compatibility

A Contract which is covariantly substitutable with another Contract is said to be *Contract compatible*. Contract compatibility is a useful term when talking about mixin rules and overrides for lists and operations. It is a concept similar to previously defined override rules – however, instead of the rules applied to individual Facet attributes, it is applied to an entire Contract.

A Contract X is compatible with Contract Y, if and only if X narrows the value space defined by Y. This means that X can narrow the set of Objects which implement Y, but never expand the set. Contract compatibility is not commutative (X is compatible with Y does not imply Y is compatible with X). Practically, this can be expressed as: X can add new URIs to Y's Contract Elements, but never take any away.

7.8 Lists and Feeds

Implementations derived from `list` or `feed` Contracts inherit the `of` attribute. Like other attributes an implementing Object can override the `of` attribute, but only if Contract compatible - a Server SHOULD include all of the URIs in the Contract's `of` attribute, but it MAY add additional ones (see Section 7.7).

Lists and Feeds also have the special ability to implicitly define the Contract of their contents. In the following example it is implied that each child element has a Contract of `/def/MissingPerson` without actually specifying the `is` attribute in each list item:

```
<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>
```

If an element in the list or Feed does specify its own `is` attribute, then it MUST be Contract compatible with the `of` attribute.

If an implementation wishes to specify that a list should contain references to a given type, then the implementation SHOULD include `obix:ref` in the `of` attribute. This MUST be the first URI in the `of` attribute. For example, to specify that a list should contain references to obix:History Objects (as opposed to inline History Objects):

```
<list name="histories" of="obix:ref obix:History"/>
```

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: list

William Cox 4/29/2015 10:38 PM
Deleted: specific

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:27 PM
Comment [9]: Here's the requirement to avoid recursion.

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: list

William Cox 4/29/2015 10:38 PM
Deleted: List

William Cox 4/29/2015 10:38 PM
Deleted: List

In many cases a Server will implement its own management of the URI scheme of the child elements of a `list`. For example, the `href` attribute of child elements may be a database key, or some other string defined by the Server when the child is added. Servers will not, in general, allow Clients to specify this URI during addition of child elements through a direct write to a list's subordinate URI.

Therefore, in order to add child elements to a list which supports Client addition of list elements, Servers MUST support adding list elements by writing to the `list` URI with an Object of a type that matches the list's Contract. Servers MUST return the written resource (including any Server-assigned `href`) upon successful completion of the write.

For example, given a `list` of `<real>` elements, and presupposing a Server-imposed URI scheme:

```
<list href="/a/b" of="obix:real" writable="true"/>
```

Writing to the list URI itself will replace the entire list if the Server supports this behavior:

WRITE /a/b

```
<list of="obix:real">
 <real name="foo" val="10.0"/>
 <real name="bar" val="20.0"/>
</list>
```

returns:

```
<list href="/a/b" of="obix:real">
 <real name="foo" href="1" val="10.0"/>
 <real name="bar" href="2" val="20.0"/>
</list>
```

Writing a single element of type `<real>` will add this element to the list.

WRITE /a/b

```
<real name="baz" val="30.0"/>
```

returns:

```
<real name="baz" href="/a/b/3" val="30.0"/>
```

while the list itself is now:

```
<list href="/a/b" of="obix:real">
 <real name="foo" href="1" val="10.0"/>
 <real name="bar" href="2" val="20.0"/>
 <real name="baz" href="3" val="30.0"/>
</list>
```

Note that if a Client has the correct URI to reference a list child element, this can still be used to modify the value of the element directly:

WRITE /a/b/3

```
<real name="baz2" val="33.0"/>
```

returns:

```
<real name="baz2" href="/a/b/3" val="33.0"/>
```

and the list has been modified to:

```
<list href="/a/b" of="obix:real">
 <real name="foo" href="1" val="10.0"/>
 <real name="bar" href="2" val="20.0"/>
 <real name="baz" href="3" val="33.0"/>
</list>
```

Page 35: [1] Deleted	William Cox	4/29/15 10:38 PM

Contract	Contracts are the templates or prototypes used as the foundation of the OBIX type system. They may contain both syntactical and semantic behaviors.