Title: Roadmap of ISO/IEC 11404 -- Its Standardization and Data Interoperability
Author: Frank Farance, Project Editor for 11404 revision (frank@farance.com)

The first edition of ISO/IEC 11404 standard, published in 1996, was titled "Information technology — Programming languages, their environments, and system software interfaces — Language-independent datatypes". Since 1996, the standard has enjoyed increasing use and applicability outside its original area of providing common semantics for datatypes across programming languages, databases, and other systems that use datatypes. The revision of 11404 incorporates new features from the technologies in the past 7 years. The standard has been re-titled "Information technology — General Purpose Datatypes (GPD)" to reflect the general-purpose nature of the standard.

This document describes the changes and their rationale for the 11404 revision.[1] The first portion discusses data and datatypes. The second portion of this document is an overview of the development of data interchange standards and how 11404 may be used to develop these kind of standards. The third portion discusses the current activities for the revision of 11404.

TABLE OF CONTENTS

PART 1: FUNDAMENTALS OF DATA AND DATATYPES
- What is Data?
- What is a Datatype?

PART 2: DATA INTERCHANGE STANDARDS
- Overview of the Consensus-Building Process
- Risk Mitigation
- De-Coupling Data Models From Bindings/Encodings
- Considering All Three: Codings, APIs, Protocols
- Extensions and Incremental Improvements
- Using ISO/IEC 11179 to Support Incremental Standardization
- Using ISO/IEC 11179 and ISO/IEC 11404 to Support Automation
- Using ISO/IEC 11404 To Support Data Interchange Standards
- Example of XML Binding Generated from 11404 Datatyping

PART 3: DEVELOPING THE REVISION OF 11404
- Charter For The Revision of ISO/IEC 11404
- Referencing, Harmonization, Mapping, and Incorporation of Standards
- For Further Information About ISO/IEC JTC1 SC22 WG11

PART 1: FUNDAMENTALS OF DATA AND DATATYPES

What Is "Data"?

According to ISO/IEC 2382-1 Information Technology — Vocabulary — Fundamental Terms, the definition of "data" is:

[1] As of this writing, the latest draft of the ISO/IEC 11404 revision is available as document WG11/N0485 at the SC22/WG11 web site at "http://std.dkuug.dk/jtc1/sc22/wg11".

> **data:** A reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or processing. Note: Data can be processed by humans or by automatic means.

Other definitions of data exist, too. A better, operational definition of "data" is:[2]

> **data:** instantiation of a relation between a concept and a sign
>
> EXAMPLE 1 The sign "…" is associated with the concept of "image in monochrome pixels" and its data is a 2-dimensional array of bits with three of the bits (representing the dots in the ellipsis) are set to black and all others set to white.
>
> EXAMPLE 2 The sign "…" is associated with the concept of "horizontal ellipsis" in ISO/IEC 10646-1 and its data is 2026 (in hexadecimal).
>
> EXAMPLE 3 The sign "…" is associated with the concept of "the letter S" in Morse Code and its data is dot-dot-dot.
>
> NOTE 1 The purpose of using the same sign in each of the examples is to show (1) how the concept influences the nature of the data (the relation between the concept and the sign), and (2) how different data can be created from the same sign.
>
> NOTE 2 The first example is a simple set of pixels (sign) that creates data consisting of a set of bits (the 2-dimensional array). The second example is a sign that is related to a permissible value (each character's code value of ISO/IEC 10646-1 is a permissible value) that is part of a value domain (ISO/IEC 10646-1 character set) whose datatype is numeric. The third example is similar to the second example except that the datatype, fundamentally, is non-numeric. Of course, for all data that is computable in a digital data processing system, it is possible to map all data to numeric datatypes.

Thus, "data" is not just a symbol (e.g., a set of bits) but must be associated with some concept.

What is a "datatype"?

In 11404, a "datatype" is defined as:

> **datatype:** set of distinct values, characterized by properties of those values, and by operations on those values

A datatype consists of three main features: a value space, a set of properties, and a set of characterizing operations.

The first main feature is: the *value space*. A value space is the collection of values for a given datatype. The value space of a given datatype can be defined in one of the following ways:[3]

— enumerated outright, or
— defined axiomatically from fundamental notions, or

[2] This definition of "data" was taken from a working draft of ISO/IEC 20944-002, Information technology — Metadata Interoperability and Bindings (MDIB) — Part 002: Common vocabulary. See "http://metadata-stds.org"

[3] This definition of value space is excerpted from 11404, subclause 6.2.

— defined as the subset of those values from some already defined value space which have a given set of properties, or
— defined as a combination of arbitrary values from some already defined value spaces by a specified construction procedure.

A new feature for the revision of 11404 is the incorporation and support of "sentinel values" (in contrast to "regular values"). A *sentinel value* is a "signaling" value, such as **`nil`**, **`NaN`** (not-a-number), **`+inf`** and **`-inf`** (infinities), and so on.

The second main feature of datatypes are: *properties*. 11404 defines six standard characteristics and their properties:

— **equality:** Is there a notion of equality among the elements of the value space?
— **order:** Is there some inherent ordering of the value space?
— **boundedness:** Is there a lowerbound, an upperbound, or both for the value space?
— **cardinality:** Is the value space finite, denumerably infinite (countable), or non-denumerably infinite (uncountable)?
— **exactness:** With respect to the computation model of the value space, are the elements of the value space the same as their conceptual values (exact), or are they just approximations (approximate)? For example, real numbers can only be approximated on digital computers.
— **numeric:** Are the values "quantities" is some mathematical number system?

The third main feature of datatypes are: *characterizing operations*. Characterizing operations are the main operations upon the values of the value space. For example, the 11404 **`integer`** datatype has the following characterizing operations:

> **Equal(x, y: integer): boolean** is **true** if **x** and **y** designate the same integer value, and **false** otherwise
>
> **Add(x,y: integer): integer** is the mathematical additive operation
>
> **Multiply(x, y: integer): integer** is the mathematical multiplicative operation
>
> **Negate(x: integer): integer** is the value **y: integer** such that **Add(x, y) = 0**
>
> **NonNegative(x: integer): boolean** is
>
> **true** if **x = 0** or **x** can be developed by one or more iterations of adding **1**,
>
> i.e. if **x = Add(1, Add(1, ... Add(1, Add(1,0)) ...)**;
>
> else **false**.
>
> **InOrder(x,y: integer): boolean = NonNegative(Add(x, Negate(y)))**
>
> **Quotient(x, y: integer): integer**, where **0 < y**, is the upperbound of the set of all integers **z** such that **Multiply(y,z) ≤ x**
>
> **Remainder(x, y: integer): integer**, where **0 ≤ x** and **0 < y**, **= Add(x, Negate(Multiply(y, Quotient(x,y))))**

Characterizing operations are needed to distinguish datatypes whose value spaces differ only in what the values are called. For example, the value spaces (**`one`**, **`two`**, **`three`**, **`four`**), (**`1`**, **`2`**, **`3`**, **`4`**), and (**`red`**, **`yellow`**, **`green`**, **`blue`**) all have four distinct values and all the names (designations) are different. But one can claim that the first two support the characterizing operation **Add()**, while the last does not.

The 11404 standard provides a rich library of datatype generators (e.g., **record**, **array**, etc.) that may be used to create complex data structures (which are supported by many programming languages). The following is a simple example of using 11404 notation:

```
type employee_record = new
record
(
       name: characterstring,  // employee name
       marital_status: state (single, married),
       exemptions: integer,    // number of exemptions for tax deduction
       pay_rates: array (0..20) of pay_rate_type,  // an array of records
),
type pay_rate_type = new
record
(
       code: characterstring, // pay code
       wage: scaled(10,4),    // hourly wages to 4 decimal digits
),
```

PART 2: DATA INTERCHANGE STANDARDS

Overview of the Consensus-Building Process

The consensus-building process can be described in four major overlapping phases:[4] *development phase*, where normative wording is formulated; *consensus-building phase*, where formal balloting and resolution occur; *maintenance phase*, where the standard is interpreted and minor refinements are incorporated; *review phase*, where the standard is revised (incorporate new technology), reaffirmed (stable technology), or withdrawn (obsolete technology).

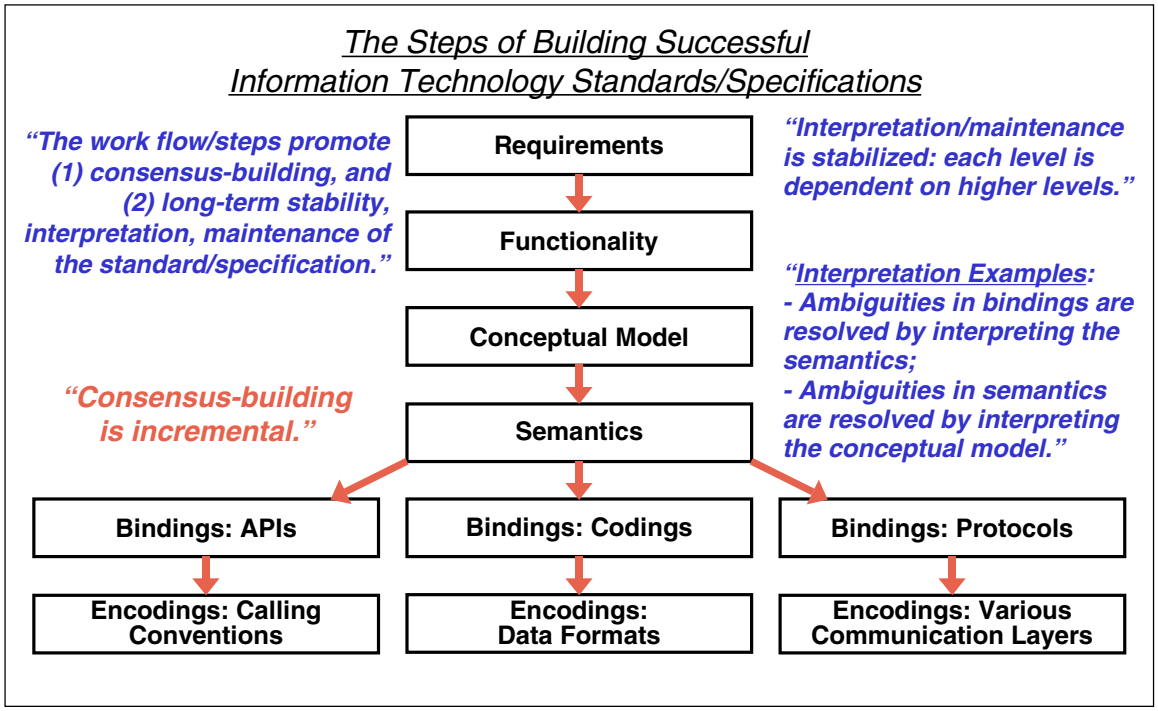
Data interchange standards use this process, too, however these kind of standards, their development, consensus-building, and maintenance involves further complications because of the ongoing fast pace of information technology standards and infrastructure.

Risk Mitigation

Standards need to be developed in short periods of time, so participants use risk mitigation strategies and methodologies. A common engineering methodology is to divide a large project into several phases of development — each phase addresses a particular category of engineering risks. For example, when building a house, it is easier to fix mistakes earlier and when on paper (blueprints), rather than waiting later after the foundation has been built. Likewise, for engineering of software or the development of standards: getting agreement on "functionality" should be done earlier rather than later. This phased approach reduces the risk of development — risk measured in terms of re-work, re-design, and re-implementation.

[4] Most standards development committees overlap these phases.

The following diagram shows a typical approach for developing data interchange standards and their bindings.[5]



In the figure above, standards development may be considered as progressing in several overlapping steps. Note: These steps are not development phases per se, but areas of development risk and technical description.

- **Requirements** allow the standard (or specification), upon completion, to be validated by reaffirming the satisfaction of the original requirements. *Result: The standard or specification is still useful. Note: Although the formal standards process is not requirements-based, the identification of requirements is a useful "best practice".*
- **Functionality** helps delimit and "contain" the scope of the standard, which minimizes "feature creep". *Result: The consensus-building process will stay focused on its technical goals.*
- **Conceptual Model** describes a virtual implementation that models the theory of operation. For the maintenance phase of the standards process, the conceptual model may be used to resolve ambiguities in semantics that were unforeseen or overlooked in the consensus-building process. *Result: The standard (or specification) can adapt to changes in technology.*
- **Semantics** are described separately from conceptual model and bindings. Semantics are not tied to or influenced by a particular binding. *Result: Semantics are binding-independent so more (future) bindings and applications are possible, thus, a longer lifetime for the standard or specification, and increased interoperability.*
- **Bindings** separate the "standard behavior" (semantics) from the mappings to particular codings, file formats, APIs, commands, protocols, transaction sets, and so on.[6] Formally,

[5] This "flow" diagram is merely a best practice. Other approaches are possible. The main point is: (1) to recognize some engineering risk and business risk is present in the development of a standard, and (2) to recognize that *one needs to choose* some strategy for mitigating risk.

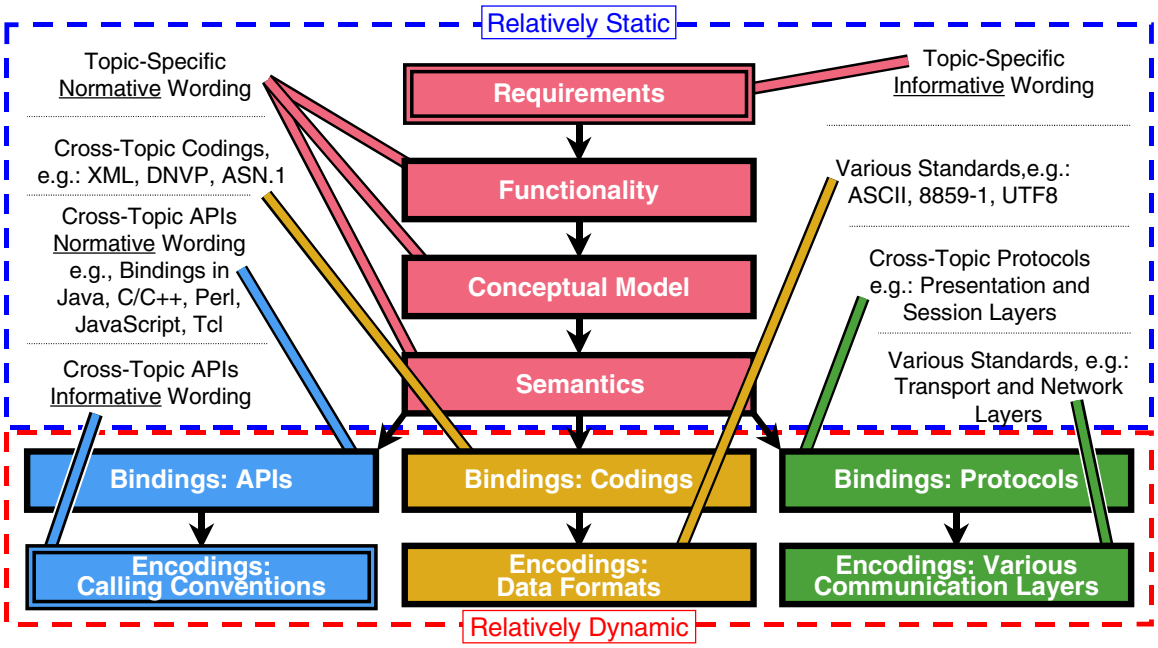
a "binding" is "a mapping from one standard or framework to another standard or framework". *Result: The standard or specification can have common functionality across many operating environments (syntaxes, file types, languages, operating systems, protocol stacks, service methods, etc.), thus the standard or specification will have wider applicability and adoption.*

- **Encodings** separate the information structure from its bit/byte representation. *Result: The standard or specification can be transformed into "native" representations that are optimal for individual, specific operating environments, thus the standard or specification will have wider adoption.*

By separating the standards or specification development into several steps, certain high-risk issues can be addressed earlier (e.g., conceptual model and conformance) while certain low-risk issues (e.g., API signatures and character sets) can be postponed.

Result: Resources are best utilized and scheduling of standards (or specification) development can be more predictable — an important management goal. This technique of using several phases of design and implementation is a common engineering methodology, whether it's software development or housing construction (better results are gained by drawing up blueprints before pouring concrete — it's cheaper to fix mistakes on paper).

De-Coupling Data Models From Bindings/Encodings



An important strategy for keeping pace with technology changes is: de-coupling the standardization of the data models from the standardization of bindings (e.g., codings, APIs,

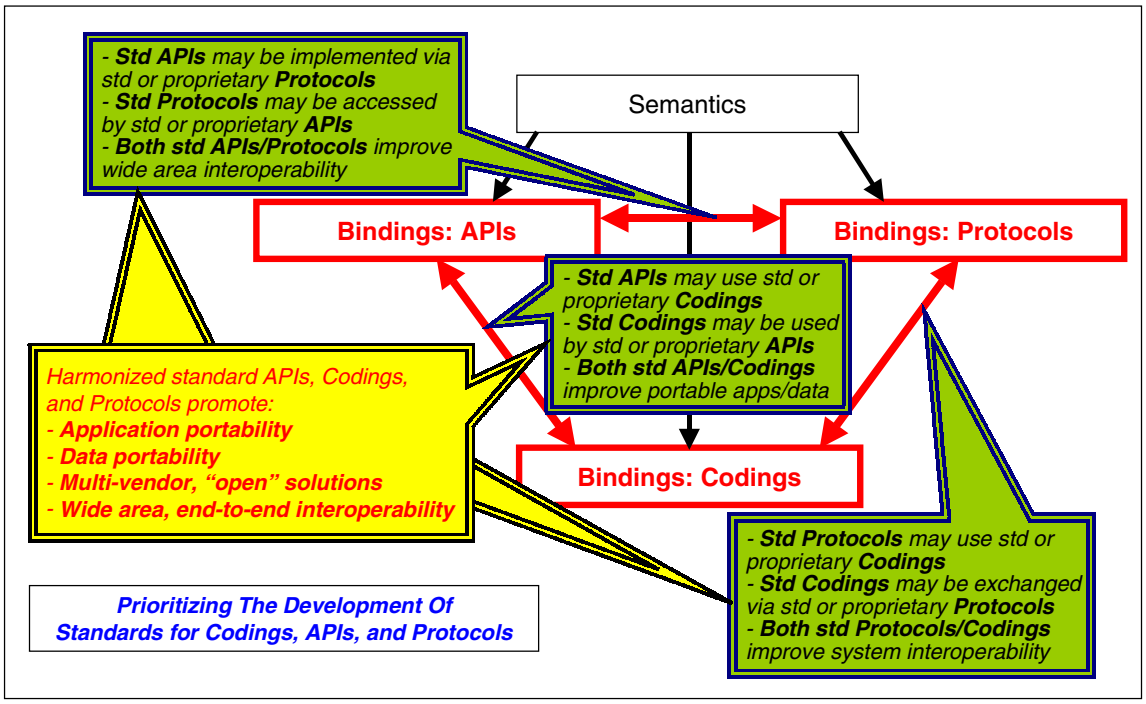
[6] The breakdown of "semantics" to coding bindings, API bindings, and protocol bindings is not as rigid as it might appear: there is no hard and firm taxonomy. For example, "remote procedure calls" (RPCs) combine features of API bindings and features of protocol bindings. Thus, the breakdown into codings, APIs, and protocols is representative of the general nature, but specific bindings may include a combination of these features or different features.

protocols). For example, these days there are many requests for "an XML-based standard for data, transactions, etc.". Translation: "I want an XML coding binding". In the late 1980's to mid-1990's, the emphasis was on API bindings, e.g., the main interoperability or standards aspect was providing, say, a C, C++, Fortran, Ada , or CORBA "binding" of services. In the late 1970's to late 1980's, the emphasis was on protocol bindings, e.g., the main interoperability or standards aspect was providing some protocol layer(s) in the ISO OSI network layering framework. Thus, since the 1980's, there have been several different ways of thinking about standards for data, information, and knowledge interchange: codings, APIs, and protocols.

Conclusion: With this kind of de-coupling, the work on the functional, conceptual model, and semantics of the standard may proceed at its own pace (e.g., a rollout of several incremental improvements via several iterations of standards revisions and amendments). Meanwhile bindings and encodings can be developed quickly, as the industry and technology demands, because the standards wording for bindings and encodings is relatively simple compared to other parts of the standard.

Considering All Three: Codings, APIs, Protocols

Typically, there aren't enough resources to standardize all the kinds of bindings that are (ultimately) necessary, so a committee may need to prioritize its standards development work. The diagram above shows a prioritization roadmap the may be used to plan a long-term roll out of data interchange standards.[7]



[7] For example, if one's highest priority is "improving system interoperability", then the primary focus should be on standardizing codings and protocols; if one's highest priority is "improving the portability of applications and

Based on the needs of the "stakeholders", standardization strategies can be chosen to meet their needs. Each stakeholder has its own set of priorities:

— **Codings:** These stakeholders consider coding standards, such as data formats, record formats, file formats, etc., as the most important aspect of standardization — these stakeholders are typically concerned about the "wire format". API standards are less important to these stakeholders because, e.g., they have no interest in building consensus around application environments or frameworks. Protocol standards are less important to these stakeholders because, e.g., they consider protocols to be merely "wrappers" for the codings.

— **APIs:** These stakeholders consider API standards, such as programming language bindings, object class libraries, command line interfaces, etc., as the most important aspect of standardization — these stakeholders are typically concerned about "portable" application frameworks. Coding standards are less important to these stakeholders because, e.g., they believe codings are implementation details that may/should be hidden. Protocol standards are less important to these stakeholders because, like codings, they may be considered implementation details that may/should be hidden.

— **Protocols:** These stakeholders consider protocols standards, such as application, presentation, and session layer services (layers 7, 6, and 5), as the most important aspect of standardization — these stakeholders are typically concerned about wide-area network interoperability. Coding standards are less important to these stakeholders because, e.g., they become transparent if there are presentation layer services. API standards are less important to these stakeholders because, for example, there are application issues "above" and outside the scope of protocol standardization.

— **Combinations:** The prioritization of pairs of binding types (e.g., codings and APIs) is discussed below. If all three binding types are important, then standards work may proceed in parallel, but the work should be closely harmonized.

Standards participants may find it very useful to consider development of all three types of bindings (codings, APIs, protocols), but it is also important to *prioritize* the work, too, i.e., only develop the bindings that are necessary now. The reason for *considering* all three even though only one is high priority is because the *consideration* of all three will ensure that future standards in the other two areas won't be hampered by design decisions at this point. Even future standards of the same binding type won't be hampered, e.g., if only an XML coding binding is desired, the "top half" (binding-independent) portion of the standard should be developed independently (but not necessarily in a separate standards document) so that it is not "tainted by XML thinking",[8] and the XML coding binding can be quickly developed — very important when one later considers developing, say, an ASN.1 coding binding.

Later on after the highest priority bindings have been developed, there will be interest in secondary priorities (e.g., if a an XML coding binding is developed first, then should an API

---

data", then the primary focused should be on standardizing codings and APIs; and if one's highest priority is on "wide area interoperability", then the primary focused should be on standardizing APIs and protocols.

[8] *Considering* multiple bindings has the same beneficial effect as developing terminology (terms, definitions) in multiple languages simultaneously: it is possible to discover and reduce the inherent bias of the working language (or the inherent bias of the primary binding for a data interchange standard).

binding or protocol binding be next?). This decision-making can be easily prioritized by observing and prioritizing the benefits of *pairs* of bindings:

- **Standardizing both Codings and API Bindings:** more portable applications and data
- **Standardizing both Codings and Protocol Bindings:** better (inter-) system interoperability
- **Standardizing both API and Protocol Bindings:** improvements in wide area application portability

At some point, all three (codings, APIs, protocols) may be standardized, which will produce the following benefits:

- Applications portability.
- Data portability.
- Multi-vendor "open" solutions.
- Wide area, end-to-end interoperability.

Conclusions: The understanding of these kind of stakeholders' priorities for the participants of the consensus-building process is most important for achieving the most efficient, shortest schedule, and highest quality standards development process.

This strategy shows how to make good progress by de-coupling the binding-independent portions of standardization from the binding (and encoding) standards. These binding (and encoding) standards can proceed in parallel. The strategy also reveals how to prioritize the standardization: (1) choose which binding type (coding, API, protocol) is most important, (2) choose which pair of binding types produce the best benefit (codings and APIs, codings and protocols, APIs and protocols) and work on them next, and (3) choose tertiary bindings as needed.

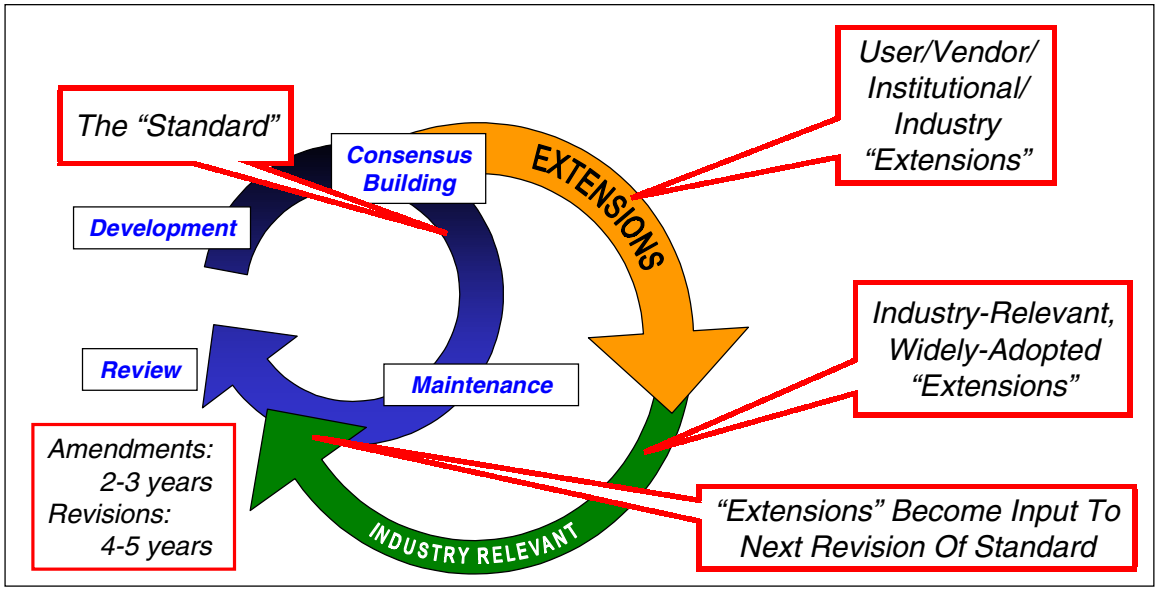
This strategy produces better engineered standards solutions over a span of **10 to 20 years**, not just the usual **3-to-5-year** lifetime of a standard.[9]

Extensions and Incremental Improvements

The standards process should not be viewed as the "be all and end all". Standards are updated and revised on a regular basis, typically every 4-5 years. Within shorter timeframes, standards are amended to incorporate new features.

Extensions are created by users, vendors, institutions, and the industry itself. Industry-relevant features can be incorporated in amendments and revisions to standards. The nature of *specific extensions*, as an extended feature of some implementation, is a conformance issue and not a technical issue per se because extensions, by their nature, are outside the scope of a standard. However, the ability to support a *general extension capability* is a technical issue and is an important consideration when developing a standard. Note that it is **impossible to support arbitrary extensions in a generic way** that provides meaning to the extensions, but it is **possible to support arbitrary extensions in a limited way**, such as (1) ignoring them or (2) retaining them but not processing them.

[9] While standards participants be primarily concerned with meeting present technical, industry, and market needs, these techniques general produce better *immediate* results and better long-term results.



Conclusions: Only the stable, agreed upon portions of technical specifications should be standardized now, while certain improvements may be deferred for future amendments and revisions. In some cases, it may be worth delaying the standards effort until more consensus can be built around improvements, but many times it can be just as important to get an intermediate or trial standard out to the public. The standards process includes "amendments" and "revisions" to support this kind of incremental delivery of a standard. Limited support of arbitrary extensions can be a significant facilitator of better interoperability and compatibility among legacy, current, and future systems.

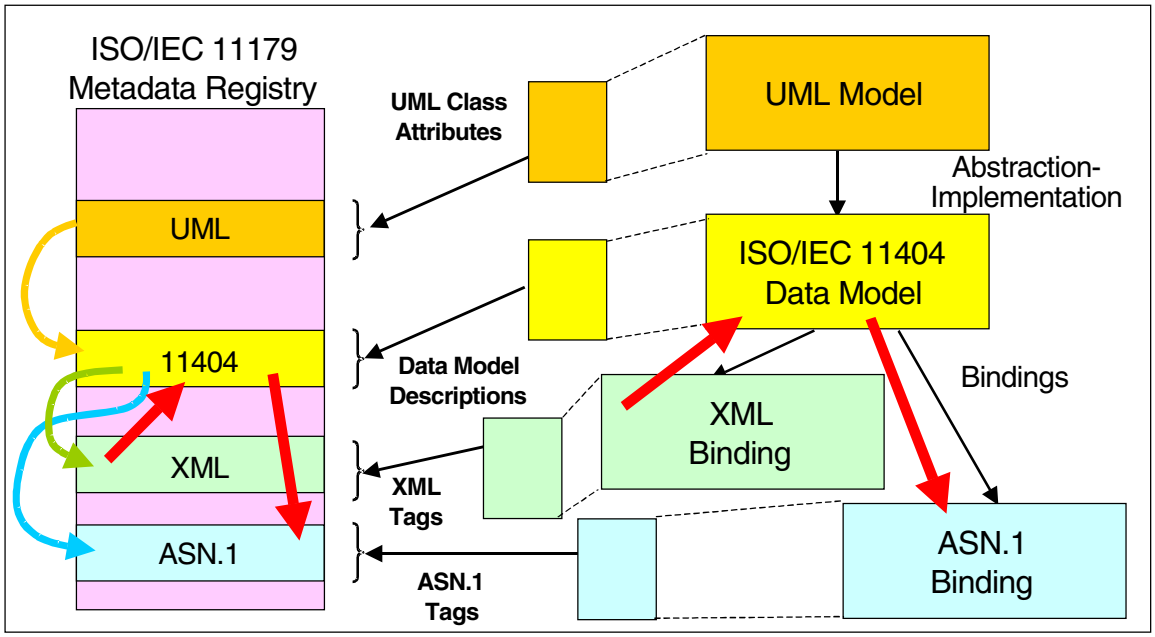
Using ISO/IEC 11179 to Support Incremental Standardization

The ISO/IEC 11179[10] standard (Metadata Registries) was developed by JTC1/SC32/WG2 (Metadata)[11] for the purpose of providing a standard method for describing data and its semantics. Registries and registration are well-known tools to support incremental standardization (e.g., the value domain ISO 3166-1 for country codes). The 11179 standard supports registration and description. In fact, for many standards an applications themselves 11404 datatyping may be combined with 11179 to support both incremental standardization and interoperable datatype specification.

[10] ISO/IEC 11179-3 specifies the "metamodel", i.e., the data model of the metadata.
[11] See the JTC1/SC32/WG2 web site at "http://metadata-stds.org". The 11179, 20943, and 20944 series of standards are freely available. See also "http://www.iso.org/ittf".

Using ISO/IEC 11179 and ISO/IEC 11404 to Support Automation

With this combination of 11179 and 11404, it is possible to provide automated translation among XML and ASN.1 data sets (assuming the XML and ASN.1 tags are registered in the 11179 metadata registry). Other automated data processing and information processing features are possible. For example: 11179 + 11404 can support a standardization effort, such as data elements and value domains are described by 11404 and further described and registered in an 11179 metadata registry that is maintained by a standards committee. As another example: 11179 + 11404 can support cross organizational data exchange by having a common language of description (11179 + 11404), regardless of the technology used for data interchange.



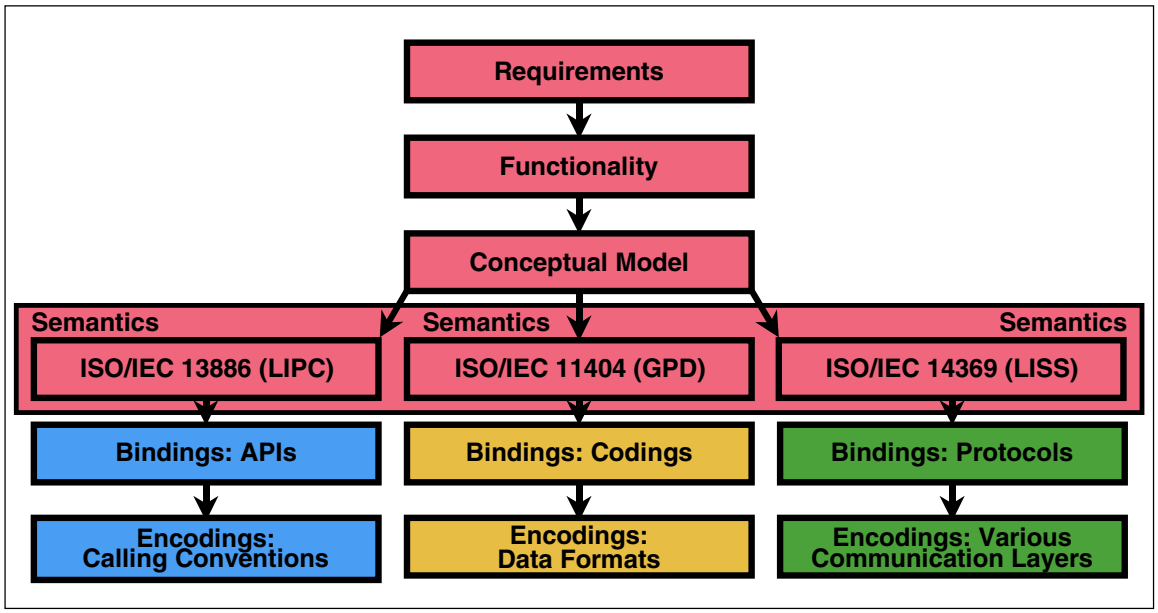
Using ISO/IEC 11404 To Support Data Interchange Standards

The "semantics" phase of development concerns the formal specification of the "meaning" of the standard. For data interchange standards, the "semantics" can be described using a variety of techniques:[12]

— ISO/IEC 11404 can be used for formal specification of datatypes
— ISO/IEC 13886 (Language Independent Procedure Calling) can be used for formal specification of APIs and, potentially, aspects of protocol specification
— ISO/IEC TR 14369 (Language Independent Service Specification) can be used as guidance for interoperability of services

The 11404 standard is particularly useful for describing datatypes in a binding independent way — not just independent of programming language, but independent of coding, API, and protocol, etc. (see diagram below).

[12] Typically, a data interchange standard would make use of some combination of 11404, 13886, and 14369.



Example of XML Binding Generated from 11404 Datatyping

A main reason for using 11404 is its precise data semantics. For example, we 11404 describes a "record", its meanings and capabilities are completely and formally defined. The 11404 standard has precise distinctions among a record, set, bag, and array; among a state and enum; among an integer, scaled, rational, and real; and so on. The 11404 standard has a rich and formal definition of commonly used (general-purpose) datatypes.

As in illustration of this formality and precision, it is possible to create a rule-based XML binding of a data model using the foundations of 11404. The following is an example of normative wording taken from an actual standard in development. This excerpt assumes that a data model has been described in the 11404 notation and that data values are to be represented and exchanged in XML.

> The following rules describe the transformation of data elements, as described by ISO/IEC xxxxx and by ISO/IEC 11404 notation, to XML records.
>
> — **Rule 1:** For each data element in ISO/IEC 11404 notation, map all identifiers to XML tags, except as noted in Rule 2 below. Represent all values in 11404 value notation. Balanced XML tags delimit the boundary of the value associated with the data element. The nesting of the XML tags represents the structure of data elements, as described by its "aggregate datatype generator" (ISO/IEC 11404 terminology). For array and sequence aggregates, (1) an XML tag of the same name as the identifier of the aggregate represents the group of aggregates, (2) the individual data elements are represented by repeated XML tags based on the identifier of the aggregate minus the suffix "_list" or "_bucket", not the index of the element.
>
> — **Rule 2:** Map all `localizedstring` datatypes[13] to:
>
> — **Rule 2A:** The locale element of `localizedstring` sets the xml:lang attribute in parent XML element.
>
> — **Rule 2B:** The string element sets content of parent tag (i.e., the current target)
>
> — **Rule 3:** Transform the following XML tags (wildcard notation):

[13] The `localizedstring` datatype is a particular feature of this illustration and not general to XML nor 11404.

`XYZ_*`

to the following XML tags (wildcard notation):

`ISO_IEC_xxxxx_XYZ_*`

All data produced shall be well-formed XML.

The normative wording above is used to describe a rule-based transformation[14] of internal data (as described by 11404 datatypes) to XML records.[15] Rule #1 serves as the main translation rule, converting the most of the data and its structures to XML. Rule #2 handles certain translation exceptions, such as transforming localized strings (e.g., `{locale="en", string="hello"}`) to XML entities and attributes (e.g., `<tag xml:lang="en">hello</tag>`). Rule #3 handles certain namespace issues[16] — in this example, namespaces are handled simply by creating unique prefixes to the tags.

For example, the rule-based XML binding could specify the translation of the following 11404 datatype to the XML records below:

```
type employee_record = new
record
(
       name: characterstring,  // employee name
       marital_status: state (single, married),
       exemptions: integer,    // number of exemptions for tax deduction
       pay_rates: array (0..20) of pay_rate_type,  // an array of records
),
type pay_rate_type = new
record
(
       code: characterstring, // pay code
       wage: scaled(10,4),    // hourly wages to 4 decimal digits
),

<employee_record>
       <name>John Doe</name>
       <marital_status>single</marital_status>
       <exemptions>1</exemptions>
       <pay_rates>
              <pay_rate><code>123</code><wage>10.0000</wage></pay_rate>
              <pay_rate><code>234</code><wage>12.5000</wage></pay_rate>
       <pay_rates>
</employee_record>
```

The reasons this kind of XML data interoperability is possible in such a short amount of normative wording is because the 11404 descriptions of datatypes is formal and precise.

[14] This illustration is based on a strategy of employing several translation phases, each with their own purpose. This illustration uses three main translation phases, but the number and type of translation phases is dependent upon the complexity of the structure of the data, and dependent upon the number and kind of exceptions.

[15] This excerpt shows the transformation from internal data to XML. For complete data interchange, the reverse operation is required: the transformation from XML to internal data. The reverse transformation is, approximately, reversing the order of the rules (e.g., Rule #1 → Rule #3) and reversing the operations, e.g., scanning the XML tags and converting them and their structure to internal data.

[16] In this illustration, namespaces are handled trivially by prepending a prefix to the XML tags. Equally trivially, XML namespaces could have been used by a slightly different re-write rule for Rule #3.

PART 3: DEVELOPING THE REVISION OF 11404

Charter For The Revision of ISO/IEC 11404

The revision of 11404 is sharply focused on achieving several technical goals. The following excerpt from the Introduction section summarizes this charter.

> **Introduction to the Second Edition (published in 200x)**
>
> This second edition incorporates recent technologies and improvements since the first edition of this International Standard. The following improvements have been incorporated into the second edition:
>
> — **Change title to reflect actual usage.** The use of this International Standard is no longer simply a tool for communicating among programming languages (old title: "Language Independent Datatypes"), this International Standard is used for formal description of conceptual datatypes in binding (or binding-independent) standards and used as formalization of metadata for data elements, data element concepts, and value domains (see ISO/IEC 11179-3). The old title was potentially misleading because readers might believe that this International Standard is only useful for programming languages. The new title "General Purpose Datatypes" captures the essence of the standard and its use.
> — **Incorporate latest technologies.** Provide enhancements to the use of ISO/IEC 11404 as a data type nomenclature reference for current programming languages, interface languages and data representation languages, specifically Java, IDL, Express, and XML.
> — **Support for semi-structured and unstructured data aggregates.** Semi-structured data and unstructured data includes aggregates where datatyping and navigation may be unknown or unspecified in advance. For example, some systems permit "discovery" (or "introspection") of data. In some cases, the datatype may be unknown in advance (e.g., a compilation time), but may be discovered and process at runtime (e.g., via datatype libraries or metadata registries).
> — **Support for data longevity, versioning, and migration.** There is a need to support, from a datatyping perspective, obsolete and reserved features, such as data elements and permissible values (enumerations and states). Marking features as "obsolete" allows processing, compilation, and runtime systems to "flag" or diagnose old (deprecated) features, while still maintaining compatibility, thus it is possible to support transitions from past to present. Similarly, marking features as "reserved" allows processing, compilation, and runtime systems to "flag" or diagnose potential incompatibilities with future systems, thus it is possible to support transitions from present to future.
> — **Extensibility of datatypes and value spaces.** There is a need to support some kind of extensibility concept. For example: (1) a GPD specification of an aggregate contains the elements A and B; (2) an application creates an aggregate with the elements A, B, and C; (3) are the application's "extensions" of the aggregate acceptable/conforming with the GPD specification in #1? The answer to #3 is dependent upon the intent and design of the specification in #1: in some cases extensions are permitted, in some cases extensions are not permitted. The extensibility concept would allow the user of GPD datatypes to describe the kind of extensions permitted. This feature is particularly important in (1) data

conformance, (2) application runtime environments that permit "discovery" or "introspection". This feature is available via the "provision()" capability.

Some features that are not incorporated within GPD are:

— **Namespace capability.** Given the larger number of declarations, a namespace capability is necessary.
— **Data representation.** Although these features are a part of GPD annotations, there is no standardization of data representation in these annotations. This step is an important link for data interoperability.

Referencing, Harmonization, Mapping, and Incorporation of Standards

In addition to this revised scoping, the following technologies, documents, and committee activities are being referenced, harmonized, mapped, and/or incorporated.

One or more of the following would be included in a **mapping annex** — alignment with datatypes in: ISO 14750 (ODP IDL); ISO 10303-11 (Express); ISO 8824 (ASN.1); XML Schema.

The following are being reviewed for **incorporation of requirements and features**: ISO/IEC 8824 and 8825 (ASN.1); ISO 14750; Open Distributed Processing -- Interface Definition Language (IDL); ISO 11303-11 and ISO CD 10303-11.2, Data modeling language Express; XML specification, XML Schema specification;[17] OMG UML, OMG XMI.

The following new or revised standards are being reviewed for **harmonization and mapping**: ISO/IEC 9899 C Programming Language; ISO/IEC 14882 C++ Programming Language; ISO/IEC 11179 Metadata Registries; ISO/IEC 20943 MDR Content Consistency; ISO/IEC 20944 Metadata Interoperability Bindings.

Liaison relationships are being developed/maintained with the following standards committees: JTC1/SC2 coded character sets, JTC1/SC6, RMODP and IDL, JTC1/SC22 programming languages, JTC1/SC25/WG4, maintenance of IEC 60559 (IEEE floating point), JTC1/SC32 data management and interchange, JTC1/SC34 document description and processing languages, TC184/SC4 product data, TC211 geographic information systems, IEEE 1596.5, scalable coherent interface (datatypes and representations). Liaison relationships are being developed/maintained with the following organizations: W3C (World Wide Web Consortium), OMG (Object Management Group), IETF (Internet Engineering Task Force).

For Further Information About ISO/IEC JTC1 SC22 WG11

Web Page: http://std.dkuug.dk/jtc1/sc22/wg11
WG11 Convener: Mr. Willem WAKKER <willemw@ace.nl>

[17] XML Schema references 11404 as a basis for developing the XML Schema concept of a datatype.

