Implementation Guide

Best Practices for Usage of DDI 3.2 and Future Versions

Increasing Interoperability and Ease of Adoption

Version 1.0, May 2017

Comments relating to the material contained in this document may be submitted to the DDI Technical Committee: ddi-srg@icpsr.umich.edu

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

This document is the Implementation Guide for the Best Practices for usage of DDI 3.2, and future DDI versions.

It has been developed to aid developers and users in the implementation of the following Best Practice areas:

- Serialization
- Identification
- Content

The following group of technical best practices offers a set of requirements and recommendations that advance both **interoperability** and **ease of adoption**.

The applicability of these Best Practices extends to all areas of DDI adoption and software production for metadata systems, including:

- Production of new metadata content profiles
- Creation of new software for use with DDI
- Updates or extensions to existing DDI software

2 OVERVIEW

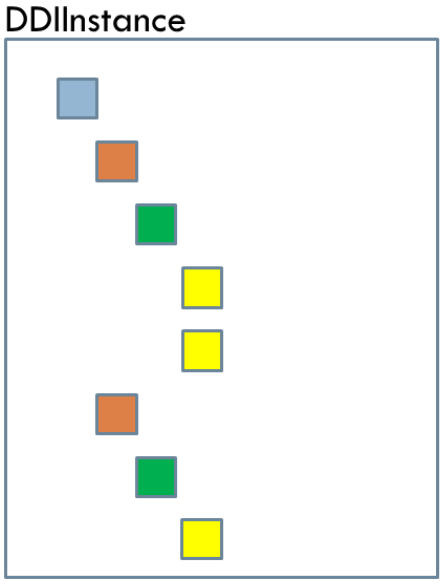
3 SERIALIZATION

3.1 DDI ITEMS SHOULD BE SERIALIZED ATOMICALLY USING THE DDI FRAGMENT INSTANCE

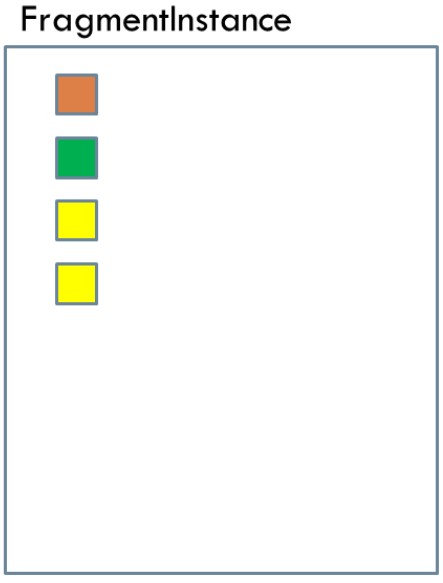
There are a multitude of improvements that became available in DDI 3.2 that combine to enable better serialization of the information model.

3.1.1 Issue being addressed

Prior to DDI 3.2, serialization of DDI required a tight coupling between the users chosen usage of the information model and the serialization. This inevitably led to interoperability problems between users and systems who organized information in differing content hierarchies. This deficiency was recognized almost immediately, and profiles were introduced as a crutch for this issue. However, these profiles do not address the main serialization issue with prior versions of the standard and still left an interoperability challenge.



Nesting mixes object model and serialization



Uniform item serialization

3.1.2 Enablers of the Solution

With the development of DDI 3.2, a model and schema checking tool was created to verify the ability to serialize all DDI items atomically using concise bounded descriptions. All locations in the DDI schema where items were nested were also detected, and these locations were updated to allow a choice of inline nesting or a reference to the DDI item. The tool also verified that all Versionable and Maintainable DDI items present within the DDI Lifecycle Model could be serialized atomically in the DDI Fragment Instance.

Identification in DDI was also decoupled from the item hierarchy in DDI 3.2. All identifiers are now scoped within an agency instead of based on their location in a DDI content model.

Item types are named uniquely as of DDI 3.2, and references now include the type of DDI item being references.

The introduction of a new serialization container, the Fragment Instance, allows for uniform item serialization. It improves locating serialized objects within an XML document since each DDI item type has only one unique xpath location, versus the unlimited number in the older serialization format. Fragment Instance, an atomically listing items with references for relationships, is the preferred XML serialization container as of DDI 3.2, and in future versions.

3.1.3 How to implement the Best Practice

- All DDI items which derive from the Maintainabe and Versionable base classes must be serialized individually within a Fragment and placed in the Fragment Instance container.
- In all locations within the XML schema where there is a choice between serializing a DDI item inline or by using a reference, the reference must be used. Usage of references creates a concise bounded description of DDI items.
- Views may be created by designating a DDI item as the TopLevelReference. For example, an Instrument could be a top-level reference when serializing a questionnaire or a PhysicalInstance could be a top-level reference when serializing a data file description. Any DDI item which can be serialized atomically in a Fragment Instance may be used as a view and have a top-level reference.
- Implementers should include all DDI items referenced by other items in the same Fragment Instance for use cases involving preservation or exchange.
- Identifiers for DDI items must be unique within the agency identifier scope to enable resolution. See 4.1.

4 IDENTIFICATION

4.1 ALL DDI ITEMS MUST HAVE IDENTIFIERS WHICH ARE UNIQUE WITHIN THEIR AGENCY

In DDI 3.2, the identification system was simplified and new implementers should use unique identifiers for all items.

Future versions also require unique identifiers.

4.1.1 Issue being addressed

Many organizations have different ways to identify their content holdings. In past DDI versions, the standard required use of a specific system of identification that was tied to the model's object hierarchy. This hierarchical identity system did not map to how many organizations managed their content. This created a very large implementation hurdle for users in adopting previous DDI 3.0 and 3.1 versions.

4.1.2 Enablers of the Solution

In DDI 3.2, the identification system was simplified. Item identifiers are now unique within the given agency identifier. Users are free to use whatever unique identification system they wish.

However, to support users who were using the hierarchical identity system, additional options were added in DDI 3.2 to support backwards compatibility of the previous serialization style and content.

Backward compatibility shims for previous 3.0 and 3.1 users

For users who have used DDI 3.0 or 3.1, and who wish to have continuity of their previous identification in DDI 3.2 and beyond, backwards compatibility was built into the simplified identification. The new identification system has added the period character as a separator for backwards compatibility. Users of 3.0 and 3.1 can translate their hierarchical identifiers into unique identifiers by concatenating the two levels of hierarchy with a single period character between the parts.

In the hope of reducing serialization changes for existing DDI 3.1 users, a scope of either Agency or Maintainable was added to identified items in DDI 3.2. When the scope is set to Maintainable on an Versionable or Identifiable item, the item's identification should be interpreted by using the items id concatenated to it parent's id with a period to create an Agency scoped identifier. This backwards compatibility shim lead to interoperability issues between implementers, so it is recommended that software creators perform this concatenation prior to serialization and record all identifiers as agency scoped and unique.

4.1.3 How to implement the Best Practice

DDI identity is composed of three parts

- an Agency / registration authority identifier (RAI)
- an Identifier / data identifier (DI)
- a Version / version identifier (VI).

These three parts combine to constitute the international registration data identifier (IRDI) in ISO/IEC 11179-6.

When assigning an identity (IRDI) to a DDI item, the identifier (DI) should be unique. Your identifier (DI) must be unique within identifiers used in your agency scope (RAI).

Agency scopes (RAI), and sub agencies, can be created and managed using the DDI Registry.

4.2 VERSION NUMBERS MAY BE TREATED AS A SERIES OF INCREASING REVISIONS

4.2.1 Issue being addressed

In DDI 3.0 and 3.1, the version of a DDI item required three levels of versioning. This became an implementation challenge during DDI ingestion for users who did not manage their content to exactly three levels.

4.2.2 Enablers of the Solution

In DDI 3.2, this requirement was eliminated. The version in DDI 3.2 now requires only a single number, with optional additional levels separated by periods.

This also allows users to treat the version as a revision counter, denoting a version change in an item without having to manage branching of item version trees.

DDI 3.2 also enables more precise branching and merging of version trees using the new BasedOn reference which is available in DDI items. This is the preferred way to handle branching and merging. See 4.3.

4.2.3 How to implement the Best Practice

Users may use a single integer to denote the version of a DDI item. Users can increment the version when a change is made to the item.

If users wish to denote a semantically meaningful version, the UserVersion of a UserId can be used. A semantically meaningful version could alternatively be included in an item's Label.

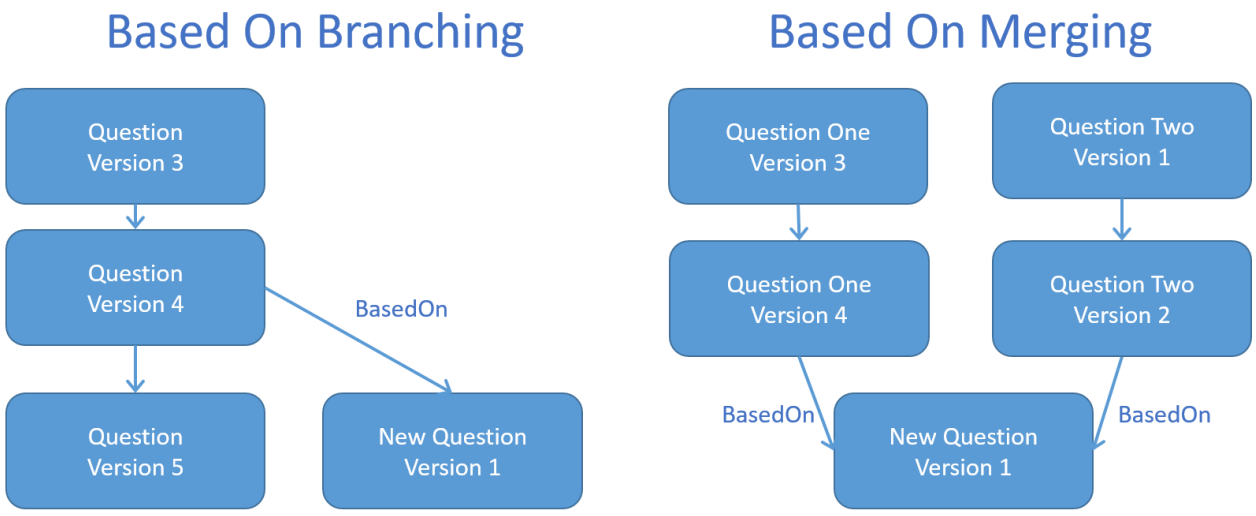
4.3 USE BASEDON FOR BRANCHING AND MERGING OF VERSIONS

4.3.1 Issue being addressed

In DDI 3.0 and 3.1, the only way to branch an item was to use multiple levels in the item's version. There was no ability to merge multiple items.

4.3.2 Enablers of the Solution

DDI 3.2 added the BasedOn reference to allow both branching and merging. Codes can also be used to describe different uses of the branching and merging.



4.3.3 How to implement the Best Practice

When creating a new item which is based on a previous item, use a BasedOn reference.

When creating a maintenance branch, or "sub version" or an item, use a BasedOn reference.

When merging multiple items, or versions of items, into a new item, use a BasedOn reference.

4.4 ISO/IEC 9834-8 UUID OBJECT IDENTIFIERS MAY BE USED FOR UNIQUE IDENTIFIERS

4.4.1 Issue being addressed

DDI requires organizations to create unique persistent identifiers (PID) to identify DDI items. Implementers have found difficulty coordinating identifier creation between their separate DDI tools or departments, leading to identifier conflicts.

UUIDs, as defined in ISO/IEC 9834-8, solve this problem.

PID minters are the alternative solution, but have several drawbacks. DDI requires a large number of identifiers, and requesting a PID from a PID minter introduces large amounts of latency. Additionally, tools created to use a certain PID minter would be more difficult to share across organizations. Finally, many PID minters, such as DOI, would be cost prohibitive for the number of identifiers DDI requires.

4.4.2 Enablers of the Solution

ISO/IEC 9834-8 defines a protocol for creating universally unique identifiers for use in distributed and registry systems. Creation of these identifiers is straight forward in all modern programming languages.

See also:

- ISO/IEC 9834-8:2014 Procedures for the operation of object identifier registration authorities -- Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers for additional information.

4.4.3 How to implement the Best Practice

When creating an identifier for a new DDI object, you may use a UUID generated by your programming platform.

Java	`import java.util.UUID;`
	
	`UUID id = UUID.randomUUID();`

C#	`Guid id = Guid.NewGuid();`

Python	`import uuid`
	
	`uuid.uuid4()`

Ruby	`require 'securerandom'`
	
	`SecureRandom.uuid`

Perl	`use Data::UUID;`
	
	`$ug = Data::UUID->new;`
	`$ug->create_str();`

Rust	`use uuid::Uuid;`
	
	`let my_uuid = Uuid::new_v4();`

When using DDI as an import or export format, Implementers may wish to store the UUID of the DDI item alongside the metadata content in their internal system.

5 CONTENT

5.1 SEQUENCES MUST BE USED WITHIN IFTHENELSE, LOOP, REPEATUNTIL, AND REPEATWHILE CONTROL CONSTRUCTS

5.1.1 Issue being addressed

The control constructs present in DDI 3.2 including the IfThenElse, Loop, RepeatUntil, and RepeatWhile allow creating branching and looping logic. They reference child items with the following properties:

- IfThenElse
  - ThenConstructReference
  - ElseIf (repeated)\ ThenConstructReference

    - ElseConstructReference
- Loop
    - ControlConstructReference
- RepeatUntil
    - UntilConstructReference
- RepeatWhile
    - WhileConstructReference

These references have in the past allowed any type of control construct derived item to be referenced. In order to simplify implementation and define scoping rules in a more precise manner, future versions of DDI will require that these properties reference a Sequence.

5.1.2 Enablers of the Solution

DDI 3.2 introduced Bindings and Parameters which can be used to reuse and compose independently created DDI items into a process model.

To improve the ease of adoption, future versions of DDI have separated control flow logic from actions such as questions and measurements. Actions will be present only within a sequence, reducing the amount of implementation work required to perform composition via bindings and parameters. This separation also reduces the amount of implementation work required to add a second item within a looping or conditional construct as a sequence container is already present for the item. The conditional and looping constructs contain only child sequences. Future versions of DDI 3.x may also restrict items referenced from looping constructs and conditionals to sequences as well.

5.1.3 How to implement the Best Practice

When creating a process model for an instrument in DDI 3.2 and beyond, use a Sequence item within all conditional and looping construct references listed above.

5.2 USE USERATTRIBUTEPAIRS FOR ADDITIONAL CONTENT NOT PRESENT IN THE DDI STANDARD

5.2.1 Issue being addressed

There are many ways to include additional content in DDI including Notes, text fields, user ids, and using items and elements for purposes not described in the documentation. This has led implementers to misuse content areas and create interoperability problems.

5.2.2 Enablers of the Solution

In DDI 3.2 and future versions, DDI items have a set of key-value pairs called UserAttributePair which should be used as an extension area. Both the key and value may optionally be controlled vocabularies, keeping user extensions managed, documented, and sharable.

5.2.3 How to implement the Best Practice

When additional content needs to be recorded with or about a DDI item, create a string to use as the AttributeKey. Store the relevant information in the AttributeValue.

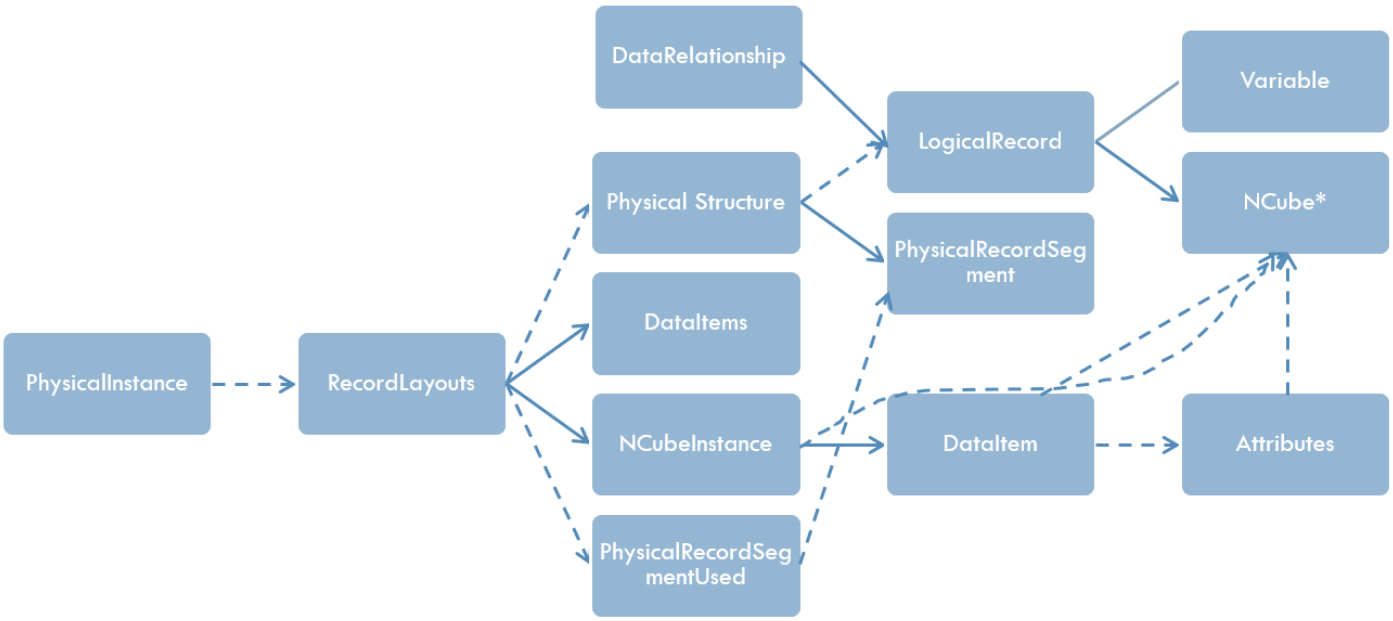
Implementers may optionally create a formal controlled vocabulary to document their attribute keys or attribute values.

It is common and implementers may use the JSON format to store information in the AttributeValue if it is multi-valued or a complex type.

5.3 USE THE DATARELATIONSHIPREFERENCE WHEN DESCRIBING A PHYSICALINSTANCE

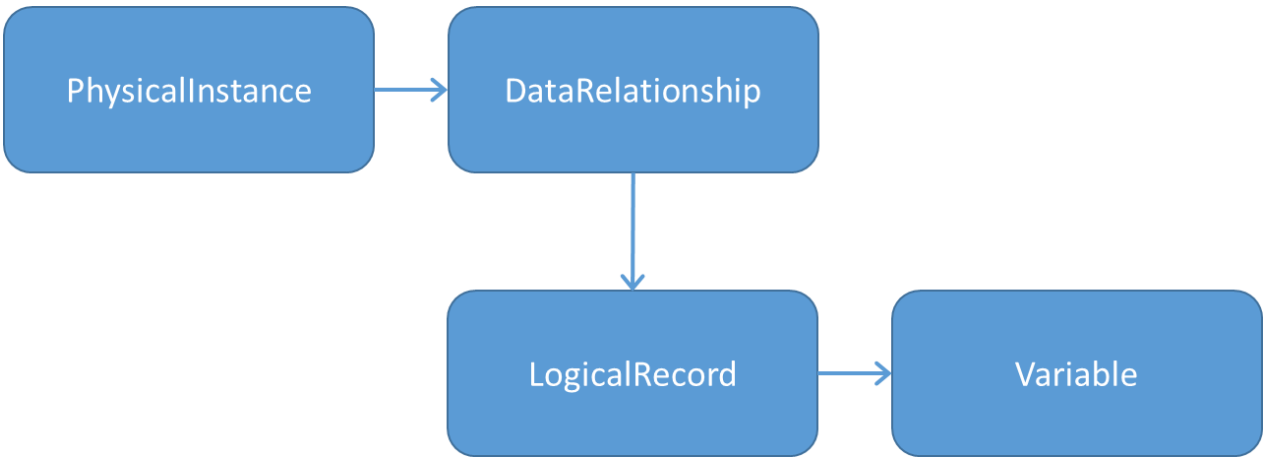
5.3.1 Issue being addressed

DDI 3.x has a very verbose system of documenting the record structures contained in physical files. This imposed a very high cost on implementers, and did not provide a way to document only the logical record of a file or dataset. The following picture details the previously required structure present in DDI 3.0 and 3.1.



5.3.2 Enablers of the Solution

In DDI 3.2, a relationship was created between the PhysicalInstance and the DataRelationship called DataRelationshipReference. This new relationship allows describing solely the logical layout of a dataset without needing to describe the on-disk format in detail. The following picture details the new simplified structure available to document a dataset in DDI 3.2.



5.3.3 How to implement the Best Practice

When creating a PhysicalInstance DDI item to describe a dataset, you should reference the DataRelationship from the PhysicalInstance and create LogicalRecords for the dataset in the DataRelationship. DDI Variables items for each variable in a record are recorded in the LogicalRecord.

5.4 LANGUAGE TAGS SHOULD BE USED ON ALL TEXT ELEMENTS

5.4.1 Issue being addressed

DDI allows multilingual text in many places throughout the model. To ensure interoperability, the language of the text should always be specified and associated directly with the text within the DDI text type that is being used.

5.4.2 Enablers of the Solution

DDI allows usage of language codes for most instances of text.

5.4.3 How to implement the Best Practice

Implementers should use the xml:lang and audienceLanguage attributes that are within the DDI standard text types. Most standard DDI text types allow repetition of text containing elements, once for each language. Each use of repeated text within a DDI text type should define the language code used.

Language codes are defined by BCP 47, Tags for Identifying Languages.

5.5 WHEN USING XHTML, AVOID USING THE SCHEMA EMBEDDED XHTML

5.5.1 Issue being addressed

DDI 3.x has traditionally embedded the XHTML schema for use in structured text. Feedback from implementers suggest that embedding XHTML within the schema makes schema based automation tools and generators fail. Validation of content also becomes required prior to serialization in DDI.

Future versions of DDI are moving to additional serialization targets other than XML. Embedded XHTML will no longer be supported in all target bindings of the DDI model.

5.5.2 Enablers of the Solution

DDI 3.2 adds an isPlainText attribute to denote if the content is encoded in some way.

Future versions of DDI will also denote what encoding is used for content, such as plain text, markdown, HTML, XHTML, LaTeX, or others.

5.5.3 How to implement the Best Practice

Plain text may be used for string content to increase interoperability and for archival preservation.

Implementers may use a structured plain text format such as Markdown, which includes some formatting capabilities. When HTML is included in markdown it should be XML encoded within the content.

All html and XHTML which is included should be treated as content of the DDI model separate from the included schema, and xml encoded. When using html or XHTML as content, the content should be XML encoded and the isPlainText attribute set to false.

5.6 AGENCY, ID, VERSION, AND URN SHOULD BE USED FOR ITEM IDENTIFICATION AND FOR ITEM REFERENCES

5.6.1 Issue being addressed

DDI 3.x allows for a choice between including a URN and including the Agency, ID, and Version of an object in items and item references. This leads to interoperability issues when implementers have expected one or the other

5.6.2 Enablers of the Solution

DDI 3.x allows implementers to specify both Agency, Id, and Version along with a DDI URN.

5.6.3 How to implement the Best Practice

When recording item identification and creating references to items, use all four available elements including Agency, Id, Version, and URN.