IHE Structured Data Capture (SDC) Technical Reference Guide (TRG)

version 0.41

Contents

1 Introduction to Structured Data Capture (SDC) 4

1.1 What is Structured Data Capture (SDC)? 4
1.2 SDC's History and Objectives 4
1.3 Organization of this Document 4

2 Overview of SDC Principles 5

2.1 SDC Design Principles 5
2.2 SDC Actors 6
2.3 The SDC Information Model: Data Entry Forms and Data Elements 7

2.3.1 Data Elements 8
2.3.2 Mapping Data Elements to SDC forms 8
2.3.3 The SDC Schema 10

3 Data Entry Forms 11

3.1 Introduction to Data Entry Forms 11
3.2 Reporting from DEFs 12

4 The Form Design File (FDF) 13

4.1 Some Important Additional Definitions for SDC FDFs 13
4.2 SDC XML Conventions 14
4.3 FDF Structural Overview 17

4.3.1 A First Look at FDF XML 17
4.3.2 Introduction to the XFCs 19

4.4 FormDesign Attributes and Properties 21

4.4.1 FDF Namespaces and the Namespaced Attributes 23
4.4.2 FormDesign Attributes 23
4.4.3 FDF and XFC Identifiers (`ID`s) 26
4.4.4 The `baseURI` Namespace 27
4.4.5 FormDesign Properties (eCC) 29

5 Introduction to SDC Basic Schema Types 30

5.1 SDC Schema File Overview (TBD) 30
5.2 Schema files. 30

5.2.1 Basic SDC Schema Type Hierarchy 30

5.3 The SDC Schema Inheritance Model and XFC Definitions 33
5.4 The BaseType (abstract) 35
5.5 The ExtensionBaseType (abstract) 37
5.5.1 The EBT **Property** Element 37
5.5.2 The EBT Comment Element 41
5.5.3 The EBT Extension Element 41
5.6 IdentifiedExtensionType (abstract): 41
5.7 DisplayedType 42
5.8 RepeatingType 43
6 The XFCs and their DEF Representations 45
6.1 The **DisplayedItem** XFC 45
6.1.1 **DisplayedItem** Substructure 45
6.1.2 **DisplayedItem** Events and Guards (TBD) 47
6.2 The Section XFC 48
6.3 The Question XFC 48
6.3.1 Question-Response (QR) 48
6.3.2 The Single Select **Question** (QS) 52
6.3.3 The Multi-Select **Question** (QM) 52
6.3.4 Capturing User Responses in **Question**s 53
6.4 The **ListField** Element 54
6.4.1 **ListField** Attributes: 54
6.4.2 **ListField** Sub-Elements 54
6.4.3 **ListField** Events 55
6.4.4 The **List** Element 55
6.4.5 The **LookupEndpoint** Element (draft) 56
6.5 The **ButtonAction** XFC (draft) 56
6.6 The **InjectForm** XFC (draft) 57
7 DEF Functional Considerations 58
7.1 The DEF Maintains and Manipulates the FDF 58
7.2 Implied Activation (IA) 58
7.2.1 Implied Activation with Complex Nesting and Untitled Sub-**Question**s 60
7.2.2 Complex Item Dependencies 61
7.3 The Untitled Question 61
7.4 The mustImplement Attribute (mI) 62
7.4.1 Optional **ListItem**s 62
7.5 Optional, Required and Conditionally-Required (CRQ) Responses 62
7.6 Conditionally Reported (CRP) Behaviors and Reporting from the DEF 63
7.6.1 The Omit When Unanswered (OWU) Model 64
7.6.2 Omit When Selected (OWS) 64
7.6.3 Use of the "?" Prefix Display Model 65
7.7 Contiguity of **ListItem** Lists 65
7.8 The Null Check Box 66

7.9 The Single Check Box 66
7.9.1 Reporting from the Single Check Box 67
7.9.2 The Locked (**`readOnly`**) QAS 67
7.10 Flavors of Unanswerable (FOU) 68
8 Rules 69
8.1 Selection Disables Children 69
8.2 Selection Deselects Siblings 70
9 Repeating Sections and Questions 71
9.1.1 Managing Repeated XFC and Component IDs with Monotonically Increasing Suffixes 71
9.1.2 Nested Repeats 73
10 DEF Validation and Reporting Results 75
10.1 Response Reporting Attributes: 75
10.2 Incomplete/Invalid DEF 75
10.3 Validating, Saving and Reporting from the DEF 75
11 Generating Reports from a DEF 76
12 The SDCPackage and its Metadata 77
12.1 The SDC Package (*draft*) 77
13 Deprecated Content 80
13.1 When are Items Deprecated? 80
14 Versioning 82
14.1 FDF Versioning (TBD) 82
14.2 Package Versioning (TBD) 82
14.3 eCC Versioning 82
14.3.1 The eCC TDVF **`version`** Format 82
14.3.2 Relationship of FDF **`version`**s to CAP Cancer Protocol (CCP) Versions 83
14.4 eCC Implementation Using IDs for FDF Items **`ID`**s and FDFs: 83
15 Data Storage, Terminologies, and Transmission 84
15.1.1 Terminology (ICD-O-3 & SNOMED CT) Maps 84
15.1.2 Data Storage 84
15.1.3 Usage Examples 85
15.1.4 Storage of Additional Metadata 86
15.1.5 Data Transmission 87
16 Converting SDC FDFs into DEFs 88
16.1 Automatic DEF Generation 88
17 Closing Comments 91

1 Introduction to Structured Data Capture (SDC)

1.1 What is Structured Data Capture (SDC)?

Structured Data Capture (SDC) is a new technology that creates interoperable, computer-readable definitions for standardized **Question/Answer Sets (QAS)** in **Data Entry Forms (DEFs)**.

SDC standardizes the creation and management of DEF QAS items throughout the data lifecycle. The SDC data lifecycle begins with the design of QAS items in DEFs and the inclusion of DEFs in DEF libraries. SDC standardizes the representation of QAS items in a DEF on a computer screen and defines the behavior of DEFs during user-DEF interactions.

After data is captured in an SDC DEF, the SDC model standardizes the transmission and redisplay of captured DEF user responses, as well as re-editing of the original DEF-captured data, and re-transmission of the DEF and its data to other recipients. SDC is also capable of creating specifications for reports based on the captured DEF data, and provides recommendations for the storage and querying of DEF-captured data.

1.2 SDC's History and Objectives

The SDC project was initiated by the Office of the National Coordinator for Health Information Technology (ONC) in early 2013 through its Standards and Interoperability (S&I) Framework initiative. Independent SDC-like technologies had emerged previously, but no technology-agnostic standard was available. SDC's technical workgroups have focused on creating standards by which interoperable forms are defined, rendered, populated and exchanged.

The SDC project was developed in cooperation with Integrating the Healthcare Enterprise (IHE), a standards organization which emphasizes the interoperability of healthcare information technology (HIT) systems, with a focus on combining constrained standards into profiles for interoperable data transmission. IHE gathers case requirements, identifies available standards, and develops technical guidelines which technical professionals can implement. IHE also hosts yearly "Connectathons" in several countries and stages "interoperability showcases" at HIMSS meetings, at which vendors assemble to demonstrate the interoperability of their products. The SDC workgroup has participated yearly in these IHE activities since 2014.

In keeping with ONC's role as a standards incubator, in April 2017, ONC transitioned the SDC project to an IHE "community led" project in which many organizations continue to evolve the work incubated by ONC. The version of SDC described in this document supersedes the October 2016 IHE SDC Profile with improvements made for the Jan 2019 US Connectathon. The SDC development team remains active under IHE and continues to add new and improved features and tools. Over time, new SDC versions will gain powerful new features. This document describes the most important features of SDC Form Design.

1.3 Organization of this Document

This document describes design of DEFs using the SDC model and touches upon several of the other data lifecycle activities. As a general organizational pattern for this document, we will introduce many topics at a high level, and then come back to them in progressively more detail.

We will begin with a brief Overview of SDC Principles. We will then cover the basic features of DEFs, the definition of some SDC terminology and the relationship of DEF features to SDC XML files (Data Entry Forms, The Form Design File (FDF)). We will then present an overview of the high-level SDC Schema structures that define SDC XML (Introduction to SDC Basic Schema Types), including some common XML patterns (e.g., Properties and Comments) that are shared by a large number of SDC elements. Next, we will look at the primary XML components that make up the bulk of SDC XML (The XFCs and their DEF Representations). This will be followed by several sections that take a deep dive into how various DEF components and behaviors are represented with SDC XML (starting with DEF Functional Considerations). We will conclude with a variety of relatively short SDC topics starting with DEF Validation and Reporting Results.

Those familiar with basic SDC principles may wish to start reading with a later chapter, such as The Form Design File (FDF) or Introduction to SDC Basic Schema Types.

2 Overview of SDC Principles

The SDC model is defined by the SDC **XML Schema**, which provides the definitions for creating XML **Form Design File**s (**FDF**s). FDFs are thus **XML instance documents** that conform to the SDC Schema definition. An FDF provides a standardized definition of QAS content and user-interaction behavior for a single DEF, and is designed to be transformed automatically into an SDC-based DEF. The QAS content inside an FDF is intended to support reusable QAS blocks called Data Elements (DEs), which are discussed later. Users' responses that are captured in the DEF are added to the FDF XML (now called an FDF-Response File [FDF-R]), and then the responses are transmitted to one or more endpoints.

2.1 SDC Design Principles

A brief review of SDC's functional and technical requirements will clarify the reason for many SDC design decisions.

- The primary use-cases for SDC are to:
  - Create interoperable clinical data-entry standards for FDFs and DEFs[1]
  - Enable downstream uses (e.g., quality assessments and public health analytics) of the captured standardized data
- FDFs, not terminologies or Common Data Elements (CDEs)[2], are the primary source of context-sensitive semantics.[3]
- SDC uses a single computer-readable information model to standardize DE content in the FDF and DEF. Thus, DEF content is standardized before[4] any user response data is captured by a DEF or data storage device.
- SDC "Form Fillers" (see below) are built to render any SDC FDF as a DEF, regardless of the FDF content.
- SDC also supports the definition of report formats, distinct from the DEF layout.
- SDC uses open-source technical standards to define technology-agnostic blueprints for DEF design.
  - SDC has no preferred programming languages.
  - SDC uses industry-standard XML-based mechanisms for the creation and interoperable exchange of FDFs and user responses.
  - SDC uses an interoperable, computer-readable, Schema-defined, XML format to represent the SDC information model, and that allows a computer to build and exchange a wide variety of standardized DEFs.

[1] The format for the FDF XML file is defined by an XML Schema. The FDF XML defines the information (e.g., questions and answer choices) that must be displayed in a computer screen (DEF), and also describes essential features of the DEF behavior when the user is interacting with the form on a computer screen.

[2] A CDE is a DE designed for widespread use and is housed in a CDE registry.

[3] In many cases, CDEs and terminologies (e.g., SNOMED CT) are insufficient to describe the full nuanced meaning of a captured response in a DEF and consideration of the DEF context is required to fully understand the captured user responses. However, annotation of FDF-DEs with CDEs and terminologies is useful in many cases, such as when aggregating data from SDC DEFs and other sources.

[4] This is a novel concept because today, data standardization is not generally attempted until data is queried or removed from a data store. At this point, the standardization process is called "data cleaning," which is a difficult task that may produce inconsistent and unreliable results.

- SDC provides an interoperability mechanism for saving and transporting user-entered data inside its original FDF, with 100% round-trip fidelity.[5]
- SDC includes XHTML support for formatting HTML-based rich text.[6]

2.2 SDC Actors

In the simplest SDC model, there are three primary software **actors** (itemized below) in the SDC ecosystem, each of which is a different kind of node in an SDC transaction network.

1. **Form Managers** (FMs) store FDFs[7] in a repository and transmit them immediately in response to requests from **Form Fillers** (FFs). FMs also have to address issues of authorization and authentication of users, generations of instance IDs, and enforcement of instance IDs and versions (covered later).
2. **Form Fillers** are software applications that:
   a. **Retrieve** an FDF file from a Form Manager
      i. Alternatively, the FF may retrieve an FDF transformed into HTML, or a URL that points to a server where the SDC HTML is hosted.
   b. **Render** the XML as a DEF using any convenient programming languages and methodologies. For example, the FDF may be rendered (transformed) to an HTML web page using the XSLT language. Users interact with the DEF inside the FF software.[8]

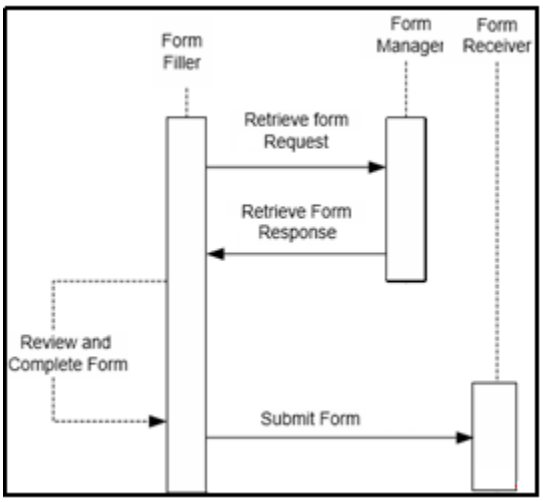


Figure 1: IHE SDC Software Actors

   c. **Capture** and validate the user-entered responses in the DEF.
   d. **Implement** implicit[9] and explicit[10] rules that define the behaviors/actions of the forms in response to user interaction.

[5] i.e., back and forth exchange of one SDC dataset (the data from an SDC DEF, inside an FDF-R) across multiple nodes, with no loss of data or context. The original DEF and its user-entered data will be visible and unchanged regardless of how many times it has been transmitted across multiple nodes.

[6] This allows the use of complex rich text (e.g., fonts, special text formatting) alongside the equivalent plain (unformatted) text in FDFs. Although many DEFs appear "better" with rich text, some systems cannot support it. Also, many data transmission standards do not natively support rich text of any sort.

[7] FMs actually store FDFs inside SDC Packages, described in The SDCPackage and its Metadata.

[8] In the case of HTML forms, the FF programming code (e.g., JavaScript) that controls the DEF behavior runs mostly inside the HTML page and the web browser.

[9] Implicit rules are DEF behaviors that are defined and controlled by the nesting structure of FDF XML.

[10] Explicit rules are predefined DEF behaviors that are specified by special SDC rule elements and attributes. The rules are based on the state of the DEF as the user interacts with it, and on the data entered into the DEF.

e. **Store** and/or **transmit** the captured response data contained inside the original FDF. As noted earlier, an FDF containing user responses is called an **FDF-R**. SDC response data is transmitted as an FDF-R to one or more actors called **Form Receivers** (FRs).[11]

3. **Form Receivers** receive the SDC response data (in an FDF-R file) from the **Form Filler** and process the data according to the FR's needs. FRs are responsible for storing the captured SDC data as native SDC XML and/or transformed into some other storage format (e.g., relational database tables). FRs may also be responsible for validation of forms via the SDC Schema, SDC rules, and external rules encoded in formats such as Schematrons. Transaction logging, validation, error reporting to the FF, version control, patient matching, authentication and authorization are other potential requirements for FRs.

Figure 1 above shows the 3 SDC actors in a sequence diagram. Much more information about FFs, FMs, and FRs may be found in the IHE Profile document, and in the supporting IHE technical files referenced therein. This document primarily describes and discusses SDC Form Filler functions.

Form Creators are the most important component of the SDC ecosystem, despite not being IHE actors (transaction nodes). Form creators are individuals and organizations, such as the College of American Pathologists (CAP), The National Cancer Institute (NCI), and Cancer Care Ontario (CCO), that define the content, structure and behavior of FDFs. Form Creators may develop FDFs by working directly with FDF XML, or by using a tool to create/edit FDF files. Form Creators send (e.g., email or upload) FDFs to one or more FMs.

2.3 The SDC Information Model: Data Entry Forms and Data Elements

We begin this section by defining a number of terms that tend to be loosely used. We will use these tighter definitions throughout this document.

DEFs are computer screens designed to capture user responses to questions that appear on the screen. They can be found in web pages, desktop business software, Electronic Health Record (EHR) forms, or any data-entry screen in any software application. **Data items** live inside DEFs. A data item is a loose concept that encapsulates a question that needs an answer (also called a "response"). In the language of SNOMED CT, a data item could be considered similar to a relatively weakly-defined[12] "observable entity." A **QAS** is a DEF data item that restricts its allowable answers to a given list of answer choices (called "list items") or to a fill-in value constrained by datatype, format, range, code system, etc.

SDC takes the QAS approach several steps further. SDC standardizes the definition of QAS items in a DEF by defining the list and/or types of acceptable responses, and then organizing multiple data items through the use of an **FDF**. It also defines the interactions between the various QAS components.

An FDF is an XML description of the data items in a DEF. It is not dependent on the programming language used to create a DEF, or the layout/design of the DEF. In other words, the FDF is technology-agnostic and primarily addresses the QAS information content and QAS-interaction behavior of a DEF.

The FDF is a highly-structured arrangement of data items, using standardized XML structures that can be read by a computer to create visible forms on a computer screen. The main purpose of an FDF is to act as a computer-readable *blueprint* for automatically creating the data items inside a DEF.

Responses (e.g., selected answer choices, or typed-in data) that are entered into a DEF by a user are said to be *captured* as data by the DEF. The computer will insert this captured data directly into the FDF in standard locations in the XML. The captured data inside the FDF can be sent (transmitted) to other locations, such as different nodes in a computer network.

[11] It is also possible to transmit data to FRs using other transport formats such as NAACCR Vol V (an HL7 2.51 format), but alternate transmission formats require more effort and are out of scope for this document.

[12] "Weakly-defined" in this context means that the domain and range of possible answers are usually not enumerated or precisely defined.

A second function of the FDF is to act as a **transport format** for the user's responses (answers) entered in the DEF. Thus FDF XML is reused to transmit the FDF-R from the F to the FR. Another function for the FDF is defining the essential metadata needed to create a formatted **report** that will be generated from the DEF responses.

The FDF transmits the user's responses by adding the responses to the FDF XML and then sending the **FDF-R** (an FDF containing user **R**esponses) to one or more receiving endpoints (Form Receivers). SDC enables *contextual*, *semantic* and *syntactic* interoperability by exchanging captured user-response data inside the FDF-R. Context refers to the hierarchical relationship between questions, answers and other form components defined by the FDF XML. Contextual semantics refers to the understanding or meaning of a question, combined with the captured answer, as affected by their position in the DEF hierarchy of QAS items. Syntax refers to the standardized structure of data items inside the FDF-R, as constrained by the SDC XML Schema.

Recipients of the FDF-R can view it as a rendered DEF, view it as a customized SDC report, or parse the data into a secondary format for some other purpose such as data analytics.

2.3.1 Data Elements

A more formal description of a data item or QAS is called a **Data Element** (DE[13]). A DE is essentially a question, attached to a definition of acceptable answer choices (either a direct "fill-in" **response** or a list of possible answer choices, called **`ListItem`s**). DEs are a fundamental concept in the SDC model. They can live in multiple information-management environments, including, e.g., inside a text document, XML, public DE registry, FDF, DEF, or database. In some cases, terminology codes and many other types of metadata may be added or mapped to DEs. When multiple DE units are nested together into an interdependent unit, the structure is known as a **complex DE**. Many FDFs contain complex DE blocks. In the SDC ecosystem, DEs are used to define the computer-readable parameters for capturing user-entered data in a DEF. The structure and metadata of the SDC DE are defined in the SDC Schema.[14]

Since DEs can live in multiple environments with different technical representations (e.g., text on paper, XML, DEFs and database models), using the term "DE" can be confusing without knowing the context. When found inside a DEF, we will use the term "**Question/Answer Set**" (QAS) for a DE. When specifically found inside an FDF, a DE will be called an **FDF-DE**. When used by itself, "DE" will refer generically to all these environments.

A **DE registry**[15] allows the sharing of DEs among many forms and allows DE variations and version history to be maintained in a central location. When found inside a DE registry, a DE will be called a **Common Data Element** (CDE).

2.3.2 Mapping Data Elements to SDC forms

SDC-embedded FDF-DE units can map to externally-hosted CDEs. As noted above, CDEs are maintained in a CDE registry, independent of FDFs. External CDEs may contain links to a variety of terminologies. This mapping could help make the DEF-captured data easier to communicate, aggregate, compare, retrieve and analyze. CDEs can be used, in principle, to construct SDC forms that share content with other SDC forms. In fact, this is the primary reason that the SDC approach is closely linked to the DE concept. While tools for CDE-based FDF/DEF design are not yet available, SDC has been architected to support this future model from first principles.

[13] Note that the letters "DE" in "DEF" do **not** stand for *data element.* DEF stands for *data-entry* form. Also, the word "Element" in "Data Element" has no intended or intrinsic relationship to an XML "element."

[14] The SDC definition of a DE is not exactly the same as that used in the ISO 11179 documentation. For our purposes, the SDC definition corrects a number of deficiencies is the ISO 11179 formulation.

[15] DE (or CDE) registries may also be called DE libraries or repositories.

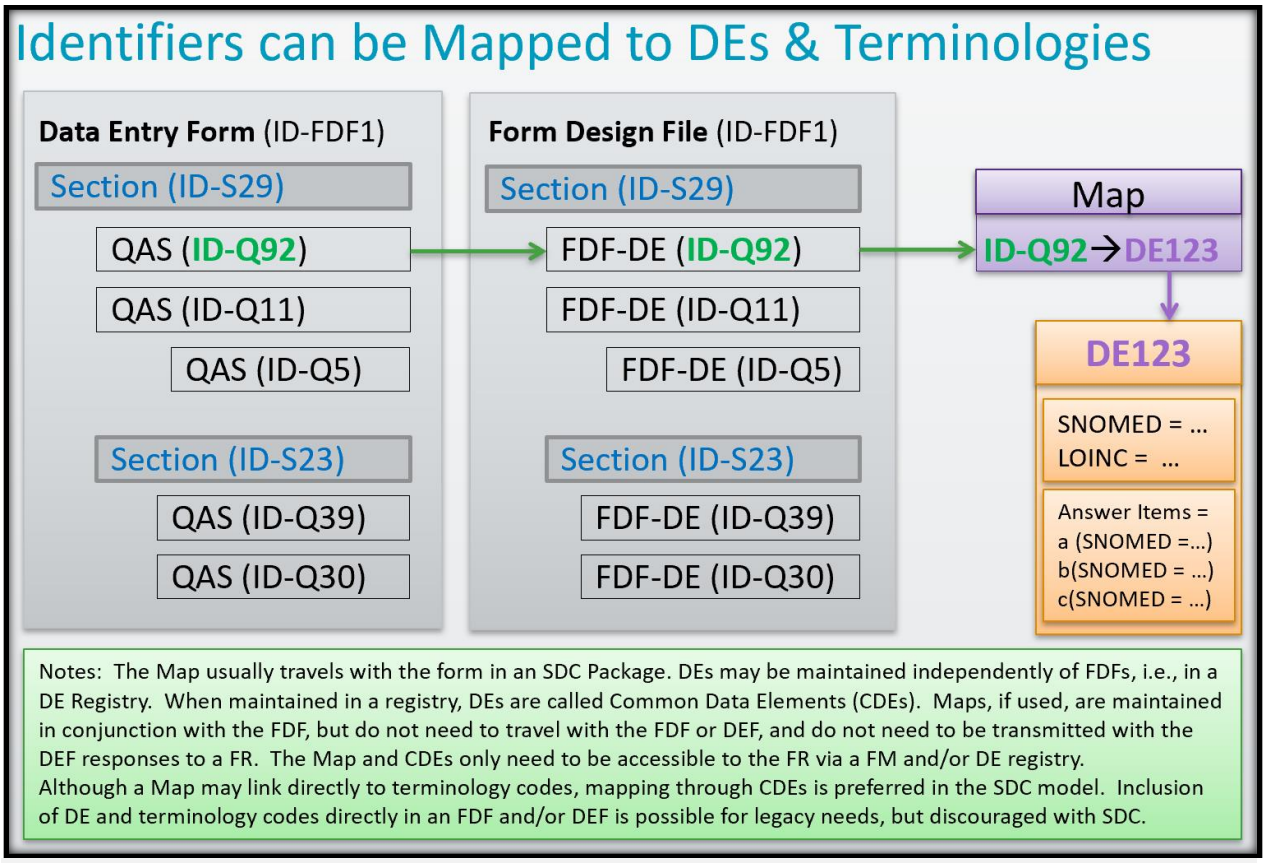


Figure 2: Identifiers can be Mapped to DEs & Terminologies

The left panel of Figure 2 shows a stylized DEF with a layered Section and QAS structure. Each QAS represents a Question. A Question may include answer choices (list items) or may accept a fill-in value, but these details are not shown in the QAS box.

Section and QAS boxes may contain child (indented) Section and QAS boxes that represent the DEF layout, and which also affect the context of the QAS responses. Child boxes generally inherit semantic context from their parent (outdented) boxes, so that the meaning of the responses in a child box is affected by its parent and higher-level ancestor boxes.

The DEF is modeled directly on the definition provided by a Form Design File (FDF), shown in the middle panel. The DEF gets its unique identifier (ID) from the FDF ID (ID-FDF1). The DEF comprises nested Section and QAS boxes that are derived directly from the Section and FDF-DE elements in the FDF. Each Section and QAS box in the DEF is associated with an ID that is imported from the FDF XML, where the IDs are defined. For example, the shared identifier of the first QAS in the DEF and FDF is **ID-Q92**.

The FDF-DE boxes without child boxes represent *simple* DEs. Section and FDF-DE boxes that have child boxes represent *complex* DEs. Complex DEs may contain complex and/or simple DEs as child items. The FDF itself may be viewed as one large complex DE. The same DE concepts apply to the corresponding DEF and its nested parts.

In the SDC model, terminology codes are ideally assigned to FDF and DEF IDs through a Map file. Each line in the Map file connects an FDF ID to a CDE, which lives in a DE registry. Each CDE is defined with an SDC DE XML structure, which may contain internal mappings to a wide variety of terminology codes and related metadata.

The Map file alternatively may link FDF IDs to terminology codes, but this is less desirable. Direct ID-terminology mapping is suboptimal because of the loss of semantic nuance resulting when the CDE structural context is not considered together with the assigned codes.

CDEs may also be mapped directly inside the FDF XML. However, this is discouraged due to the maintenance burden that arises whenever CDE mappings change.

Least desirable is the mapping of terminology codes directly in the FDF XML. This approach incurs an extremely high maintenance burden. It is only acceptable for "quick and dirty" use cases, DEFs that are not expected to be maintained over extended periods, or for DEFs and code sets that are extremely stable over time.

When mapping terminology codes to a CDE inside a DE registry, the SDC **CodedValue** element is used inside the SDC DE XML. When adding CDE or terminology codes directly inside the FDF XML, the identical CodedValue element structure is used inside the FDF XML

The transmission of user responses from a DEF involves the transfer of FDF IDs. Ideally, no CDE or terminology codes will be transmitted. Instead the FR can use the Map file to look up the appropriate CDE or terminology codes from the mapped FDF IDs. Nevertheless, SDC does support the transmission of mapped CDE and terminology codes when required for legacy use cases. In general, this practice should be phased out, and the SDC Map model used instead, whenever possible.

A further discussion of DEs, CDE registries and terminologies is beyond the present scope of this document.

2.3.3 The SDC Schema

The SDC Schema expresses an ***information model***[16] that defines the permitted FDF XML structures. Rather than have one XML Schema for each FDF, SDC uses a single XML Schema that can be used to define an infinite number of different FDFs. The SDC Schema defines the FDF structure using nested DE units, and also provides a rich array of ancillary metadata templates suitable for defining any type of DEF QAS content and behavior. The SDC Schema files are heavily-annotated and are supported by extensive HTML-based documentation.

[16] For our purposes, an information model is a logical design, expressed e.g., as an XML Schema or UML diagram, that is used to model generic information (e.g., Data Elements, Ontologies), rather than specific domains of information (e.g., medications or car parts). The SDC information model uses the generic DE paradigm to represent any information domain.

3 Data Entry Forms

3.1 Introduction to Data Entry Forms

Each FDF and DEF contains six main types of components: **Section (S)**, **Question**[17] **(Q)**, **ListItem (LI)** (i.e., answer choices[18]), **Displayed Item (DI)** (e.g., notes and other screen text), **ButtonAction (BA)**, and **InjectForm (IF).** The XML FDF components have visible counterparts in the DEF. Q and LI components implement DEs in the DEF. The other components serve mainly to support the DEs, DEF organization and DEF behavior. Each component can store additional hidden metadata which help describe the layout and behavior of the DEF and the resultant report produced from the DEF.

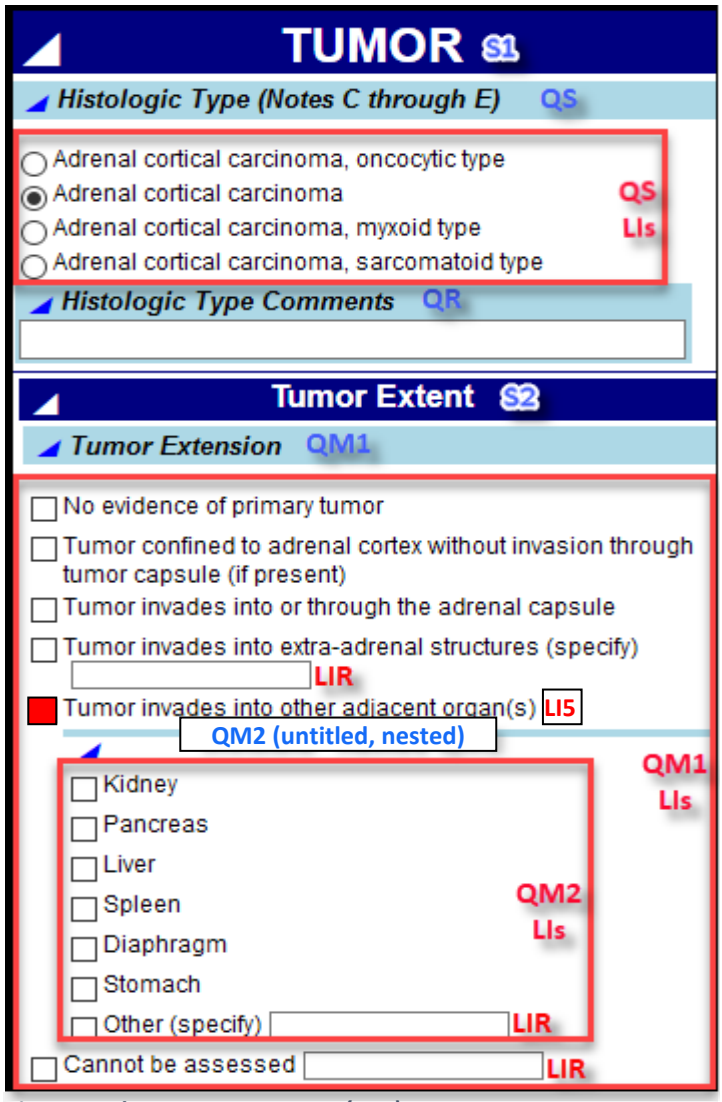


Figure 3: The Data-Entry Form (DEF)

Figure 3 shows a small section from a single FDF, displayed using an HTML DEF. The figure demonstrates S, Q and LI components only. BA and IF items are not shown. Locate each of the following items in the figure:

A thick **dark blue bar** is used for Sections, and **light blue bars** are used for Questions. ListItems (answer lists, displayed with radio/option buttons or check boxes) appear indented beneath the Question bars. Blue or white triangular icons (◢) are used to expand and collapse various parts of the DEF.

Observe that all LIs are children of a parent Question. The **red boxes** in the figure mark the *List* area for each Question, and these boxes contain multiple **LI**s. The **LI**s for a multi-select Question are represented here as checkboxes, and single-select LIs are represented as radio/option buttons (circles).

Some of the LIs are marked as a **ListItemResponse** (**LIR**) component, which is an **LI** with a special *Response* area (a thin, empty rectangular box in the figure) that can receive user-entered text after the **LIR** is selected or checked.

S1 and **S2** are Sections that contains other types of components. **S1** contains 2 Questions, **QS** and **QR**, and also contains Section **S2**. **S2** contains the Question labeled **QM1**. Containment is indicated by the slight indentation of the child items under the parent item.

Questions (Q) exist in 3 main subtypes: **QR**, **QS** and **QM**. A **QR** is a Question that can capture a fill-in or **response** value. The **QR** is displayed with a **light blue title** followed below by a thin rectangular box representing the user-response area. **QS** is a single-select Question, and **QM1** and **QM2** are multi-select Questions. The last Question,

[17] Reference to formal SDC components will henceforth be represented with capitalized terms, e.g., Question, Section, etc.

[18] The terms "ListItem" and "answer choice" are used interchangeably in this document. ListItem refers to the SDC element name for an answer choice in the DEF.

QM2, is nested under a **LI** (labeled **LI5**)of its parent Question. **LI5** is checked, as illustrated with a filled red checkbox. **QM2**, nested under **LI5**, has no title text in the thin, **light blue title bar area**, making it an "untitled Question" (sometimes also called an "invisible Question").

The component nesting structure is critical as it controls the activation of QAS items in the DEF. For example, the untitled question **QM2** and the **QM2 LIs** are only *activated* when the parent **LI** (**LI5**) is selected. Conversely, if the **QM2 LIs** are activated, then we have an implied rule that the parent **LI5** must be selected. The term *activated* means that the DEF user can respond to the on-screen Question **QM2** by selecting one or more of the **QM2 LIs** as an answer choice. The nesting structure therefore implies some significant rule-based DEF activation *behavior*, and as we shall see later, this behavior precisely follows the layout and metadata of the FDF XML.

This was a first introduction to SDC jargon for basic DEF features. We will return to all of these observations in more detail, later in this document.

3.2 Reporting from DEFs

Before diving deeper into FDF details, we need to briefly introduce the concept of **report generation** from DEFs. FDF-based reports can be tailored for target audiences, e.g., for patients versus physician specialists, or tailored to produce short summaries or aggregates of data from multiple sources.

The DEF wording and layout can be used as a guide to design reports; however, this can result in suboptimal report readability. An important issue in report output is that optimal text and layout of Questions and ListItems on the report is not always the same as optimal text and layout in the DEF. The *text* preferred for reports is represented in the SDC metadata using a special Property element feature called *reportText*, to be described later. The *layout* difference between the report and the DEF is highly dependent upon local standards and personal preference, making it difficult to completely specify an optimal layout in an FDF. However, some aspects of the preferred report layout, such as the nesting of data items and display of numeric units, can also be specified using SDC metadata in the FDF. In some cases, FDF metadata are critical to ensure proper rendering and interpretation of reports. We will return to some of these topics at various points later in this guide.

4 The Form Design File (FDF)

4.1 Some Important Additional Definitions for SDC FDFs

XML Form Component (XFC): The parts of FDF XML that are used to represent primary DEF controls (see "Control" below) are called *XML Form Components.* Similar to the prior DEF example, XFCs may be one of several types: **`Section`** **(**S**)**, **`Question`** **(**Q**)**, **`ListItem`** **(**LI**)**, **`DisplayedItem`** **(**DI**)**, **`InjectForm`** **(**IF**)**, and **`ButtonAction`** **(**BA**)**. This document will focus primarily on the first 4 XFCs listed above.

Each XFC has a unique identifier attribute (`ID`)[19]. An XFC may also have several other XML attributes and sub-elements specific to a particular XFC type. The `ID` attribute allows the unique identification of XFCs inside a DEF and also identifies DEF user responses when they are stored as captured data in a database. Every FDF-DE is composed of one or more XFCs. XFCs exist only in FDF XML, but they define the visible items (*controls* or *widgets* – see below) that appear in a rendered DEF.

Question/Answer Set (QAS): As noted above, a QAS is a **`Question`** in a DEF that includes a definition of its permitted answer(s). QAS items in an SDC DEF are created from XFC "blueprints" present in the FDF. The permitted QAS answers may be *captured* as a *fill-in response* by the user, or by *selecting* from a list of answer choices (**`ListItem`** elements). A QAS can be thought of as a DE that lives in a DEF rather than in an FDF or CDE registry.

Control: XFCs can be rendered in a DEF in different ways depending on a Form Filler's programming technique and visual themes/preferences, but all SDC forms use the same SDC information model and XFC-derived form components. When XFCs are rendered in a DEF, the DEF screen objects are called *controls* or *widgets* to distinguish these rendered screen items from the XFCs described by the FDF XML. The selection of appropriate controls for each XFC type is part of the art of programming, and FF designers are free to innovate in this area.

Metadata: In this document, the word "metadata" refers to the values of all FDF attributes which affect the rendering, behavior, storage, exchange, reporting and interpretation of FDF form components.

Data: In general, we will use the word "**data**" (as distinct from metadata) to refer to the user-entered responses captured in the SDC DEF. User-entered data are also described as "*captured"* user responses. Captured responses (data) arise in two main ways: they may be *selected* from a list of **`ListItem`**s, and/or entered directly as a response in a DEF response field. A directly entered "fill-in" response has an associated data type (e.g., string, integer, binary, etc.) and additional attributes that restrict the valid entries. The FF should validate fill-in (response) data before they are submitted to a FR. Some data may be set as **default** values and may also be marked as **read-only** (or "locked") in the FDF.

Context refers to the hierarchical nesting relationship between form components defined by the FDF XML, and the effect of this relationship on the meaning and interpretation of the nested DE components . Context (especially ancestors) can greatly affect the semantics of the DE units in an FDF and DEF. Incorrectly separating the DEF user responses into separate data slots (e.g., "shredding" captured QAS responses into database tables and fields not designed for SDC) can result in loss of data integrity and could lead to erroneous conclusions. Care must be taken to assure the preservation of the original context. Context is especially important to consider when storing DEF responses and assigning terminologies/codes. Preservation of context is a major advantage of SDC over other technologies. Terminologies (e.g., SNOMED CT codes) are not a reliable substitute for SDC context preservation.

[19] In the SDC model, each `ID` must be unique inside its form. However, it's possible for the exact same `ID` to be found in a different form, identifying a completely different object; this repeat use of `ID`s is strongly discouraged, but may be required in some legacy use cases. When a form is versioned, e.g., from version 1 to version 2, the `ID`s of the XFCs are generally preserved, unless the XFC changes significantly. The management of XFC changes, including the determination of significant modifications that force an `ID` change in a versioned form, will be discussed in detail in Versioning.

4.2 SDC XML Conventions

This section introduces SDC XML, and we assume the reader has basic familiarity with XML and XML Schema. In addition, the reader should become familiar with a few conventions specific to this document and to SDC:

- XML elements are capitalized with PascalCase, bolded, and use this style: **MyElement.** However, W3C datatype-derived elements like **dateTime** use camelCase.
- Sometimes we will use pluralized SDC **Element**s. The ending letter "s" will not use the special format for **Element**s. Note that the "s" in "**Element**s" uses the regular paragraph font.
- Attributes almost always use camelCase (except for **ID**), and they use this font: myAttribute. When used in-line with text, the style is **bolded** for added emphasis.
- **Referring to the *content* of an attribute:** The stuff between the 2 quotes after an attribute will be called attribute *content*: For example, **myAttribute** ="my content". Note the special content font. Often, the attribute content is bolded for emphasis: "**my content**".
  - We will avoid referring to an attribute's *value*, because the term *value* has a more-specific meaning in SDC, as we'll describe in the bullets below.
  - Sometimes we will call the stuff between the quotes *content value,* especially when referring to numeric content.
  - Sometimes, we will refer to an attribute's content by the attribute name itself. For example, we may write: "If the **title** is 'My Question' ", rather than "If the content of **title** is 'My Question' " or we might write "When **minCard** > 0, …".
- **Attribute-centric content:** SDC XML elements do not hold text content directly (i.e., between the opening and closing XML element tags).[20,21] All SDC XML *element content* is contained in *attribute content*, between attribute quotes. The *definitive* element content (which ordinarily would be text between the opening and closing element tags) is instead placed in a **val** (value) attribute when appropriate. This XML formatting restriction allows for simpler and leaner XML structures, supports cleaner extensibility using sub-elements, and simplifies the automated generation of programming code to create an SDC object model.
- **Documenting attribute features:** When we describe attributes, we often need to describe optionality, default values for missing attributes and the datatype for attribute content. Optional attributes are documented with [O]; Otherwise they are required and must appear in the XML. Default values are shown as [def: *value*]; Attributes missing in the XML assume the default value. Datatypes are shown as [dt: *type*].
- A named **Property** type is a custom SDC **Property** element that is not defined explicitly in the SDC Schema. (See The EBT Property Element for examples.) It is used frequently to allow the introduction of named custom metadata into FDFs through the use of the SDC **Property** element. Named **Property** types use this shorthand style: **myProp Property**., where ***myProp*** is the name of the **Property**.
  - The "***myProp***" expression is actually the content of the **propName** attribute on the **Property** element.
  - In proper XML format, the ***myProp*** **Property** looks like this: **<Property propName="*myProp*"/>**
  - Why is this convention useful? Because speaking or writing the expression "the ***myProp*** **Property**" is a lot easier than reading the XML string like this: "*The* ***Property*** *pattern in which the* **propName** *attribute content is set to* ***myProp***."

[20] It is possible to bypass this restriction when using an SDC **XML** or **HTML** element, or when using a custom SDC **Extension**. Use of these exceptions requires the declaration of an external namespace and an XML Schema, e.g., for using XHTML to record richly-formatted text inside an FDF.

[21] This applies to both data and metadata, and also applies to all SDC datatype elements which follow W3C conventions.

- When we refer to the *value* of an SDC element, we really mean the content of the `val` attribute on that element (assuming it has a `val` attribute). Thus, this statement, " the value of the `myProp` **`Property`** is "`my Val`" " means: `<Property propName="myProp" val="my Val"/>`
    - To avoid confusion with the `val` attribute, we do not refer to the *value* of an attribute. Instead, we will refer to the *content* of an attribute.

All of these styles use Courier New font, 9 point. Occasionally we will change the font size, color etc., to adjust for space constraints or for special emphasis.

Comments: To display XML comments inside attribute lists as well as between elements, `we use this style`.

The following Example shows the general structure of SDC XML:

```
<ElementTag val="ElementTag content goes here">
SDC does not allow element content here, between the opening and closing element tags
    <SubElementTag myAttribute="SubElementTag content goes here"/>
SDC does not allow element content here, between the opening and closing element tags
</ElementTag>

<ElementTag val="ElementTag content goes here"/>use of val allows more concise XML
```

Example 1: Attribute-centric XML content

XPath notation: This document will occasionally use XPath notation to concisely indicate nested elements and attributes. For example, `Question/ListField/List/ListItem/@ID` refers to the content (`"LI1a"`) of the `ID` attribute at the end of the highlighted XML path. (In the example, non-essential attributes are omitted for clarity.)

```
<Question>
    <ListField>
        <List>
            <ListItem ID="LI1a"/>
            <ListItem />
        </List>
    </ListField>
</Question>
```

Example 2: XML to demonstrate Xpath notation

Schema Types: All SDC elements are modeled with an XML Schema complex Type. For example, a `Question` element is based on an XML Schema Type called QuestionType. Elements may be discussed by reference to the element itself or its Schema Type.

The word "Schema" is capitalized to indicate that we are working with a W3C Schema, and not some other kind of custom schema system. Similarly, all SDC Schema Type names end with a capitalized "Type" suffix[22] to help distinguish the Schema Type from the XML element that derives from it.

SDC Datatypes: Almost all SDC datatypes are modeled using the XML Schema datatype model. Thus, every relevant XML Schema datatype is recreated using a new SDC Schema Type modeled after its XML Schema datatype definition. For example, the W3C integer datatype is recreated in the SDC Schema with an SDC datatype called *integer_DEtype*. The suffix DEtype is a convention to indicate that this SDC Type is used for **D**ata **E**ntry (i.e. to capture user response data) by the DEF user. Each SDC datatype also has a "**S**imple" type, such as *integer_Stype*. The difference between DEtype and Stype datatype attributes is the inclusion of data entry restriction attributes (e.g. `maxExclusive`, `maxInclusive`, `totalDigits`) in the DEtype to constrain what values the user may enter in a DEF. Stype datatypes do not have the DE restrictions. Stype datatypes are used occasionally by form designers to hard-code data into forms in parts of the SDC XML that are not subject to data entry by users and thus they have no need for data entry restrictions. For example, a form designer may add a coded integer value (e.g.,

[22] The only exceptions to the capitalization rule are SDC type names ending with “DEtype and ”Stype”

10) inside the SDC XML to annotate a clockface position of a selectable **`ListItem`** that corresponds to the 10 o'clock position. This coded value would be entered by the form designer using an XML element defined by integer_Stype.

SDC has two datatypes not defined by XML Schema: XML and HTML; both of these exist as DEtype and Stype. These datatypes allow users and form designers to enter custom XML and XHTML at various places in SDC forms. Both datatypes require the inclusion of the appropriate namespace in the SDC XML. To validate XHTML in an FDF, the xhtml.xsd Schema must be included, and to validate custom XML, a custom Schema must be provided for that non-SDC XML.

4.3 FDF Structural Overview

4.3.1 A First Look at FDF XML

FDF documents adhere to a basic layout, as shown in Figure 4.

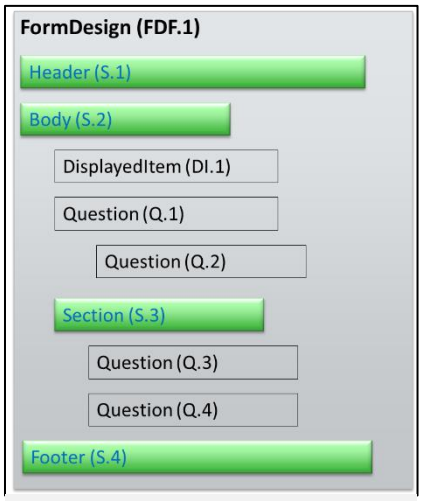


Figure 4: Basic FDF Layout

FormDesign and XFC element blocks contains a unique **ID**, which is shown in parentheses in the figure. The top **FormDesign** (**ID** FDF.1) wrapper element subsumes the top-level sections called **Header**, **Body** and **Footer**. Each section is depicted as a raised green rectangle, to emphasize the role of sections in organizing the FDF and DEF. Inside **Body** are a **DisplayedItem** (acting as text on a DEF, **ID** DI.1), a **Question** (**ID** Q.1) that subsumes a second **Question** (**ID** Q.2), and a **Section** (**ID** S.3) that subsumes 2 other **Question**s (**ID** Q.3 and **ID** Q.4). The detailed structure of the elements is not shown, but the hierarchical nature of the form layout should be clear.

Please refer to the SDCFormDesign Schema and sample SDC XML instance documents when studying this section. Note that some Schema parts, especially those related to SDC rules, may not be covered in this document.

We will now look at a simplified FDF example to study some basic features of the FDF XML layout.

```
<FormDesign

Namespace attributes go here:
    xmlns="urn:ihe:qrph:sdc:2016"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

Optional Location of the SDCFormDesign Schema, used for FDF validation:
    xsi:schemaLocation="urn:ihe:qrph:sdc:2016 file:SDCFormDesign.xsd"

SDC FormDesign attributes go here:
    order="0"
    baseURI="sdc.org"
    lineage="Samples.Samp1"
    version="v001"
    ID="Samples.Samp1_v001_sdcFDF"
    fullURI=
"_baseURI=sdc.org&amp;_lineage=Samples.Samp1&amp;_version=v001&amp;_docType=sdcFDF"
    filename="Samples.Samp1_v001_sdcFDF.xml"
    formTitle="Sample Blocks"

Custom FormDesign Properties go here:
    <Property propName="ShortName" val="my.SDC.Form"/>
    <Property propName="ApprovalStatus val="CTP1""

Main FormDesign content goes here:
  <Header ID="H1" title="This is a Header section (optional)"/>
  <Body ID="B1 title="This is the main Body section (required)"/>
  <Footer ID="F1" title="This is a Footer section (optional)"/>


    <Rules/>
</FormDesign>
```

Example 3: Top-level FDF structure

Observe the following basic points about the SDC XML in Example 3:

- The top (root) node of an FDF is **FormDesign**. Listed under **FormDesign** are many XML attributes, which provide important metadata to support **FormDesign**.

- The attributes may occur in any order. Most of the possible attributes are shown in Example 3. Optional attributes only appear in the XML if they contain a value. If an attribute holds its default value as defined in the SDC Schema, then it may not appear in the XML, and is usually omitted to conserve space.
- The FormDesign attributes are divided into three types:
    - Namespace-related attributes, beginning with "xmlns".
    - An optional **schemaLocation** attribute that describes the location of the SDCFormDesign Schema, which is useful for FDF validation.
    - Regular SDC attributes: we will cover these later in detail
- **FormDesign** is not an XFC, but like an XFC, it does contain a required **ID** attribute. It also contains several other required **ID**-related attributes specific to the FDF root. These will be described later in detail (see FormDesign Attributes and Properties).
- In addition, **FormDesign** may contain any number of standard or custom **Property** elements (see The EBT Property Element) to provide essential metadata about the form.
- **FormDesign** wraps three important nodes: **Header**, **Body** and **Footer**. These nodes may appear only once in the FDF, always under **FormDesign**. These three nodes are just top-level **Section** XFC nodes that have special descriptive names because they have specialized functions.
    - When the FDF is rendered as a DEF, the **Header** section "sticks" to the top of the DEF, the **Footer** section sticks to the bottom of the DEF, and the **Body** section is the main middle section of the DEF. The word “sticks” means that the Header and Footer do not scroll off the page when the user scrolls through the Body Section of the SDC DEF. The **Body** section may be scrolled by the user, but the **Header** and **Footer** must always appear on the DEF screen.
    - The **Body** section is *required* in the FDF, but **Header** and **Footer** are optional. The **Header**, **Body** and **Footer** may contain nested XFCs (including other **Section**s), and this nesting may continue to any depth.
    - The optional **title** attribute on each of these elements contains the text that will appear in the DEF.
    - Since each of these three nodes is a kind of **Section** XFC, they each require a unique **ID** value.
- The **Rules** element at the bottom holds advanced instructions that control some of the behavior of an FDF as the user interacts with XFCs in the form.
    - Many FDF behavioral instructions can also be attached directly to the XFCs, and do not appear under **Rules**. The instructions that appear here are mainly those that affect interactions between multiple XFCs that can be located in widely-separated areas of the FDF’s XML tree. This document does not currently cover any of these types of advanced SDC rules.

Each of the SDC elements and attributes will be explained later in more detail (see Introduction to SDC Basic Schema Types and Introduction to the XFCs).[23]

[23] In addition to the displayed FormDesign parts, the **FormDesign** element may contain custom **Extension**s, **Comment**s, and events (**BeforeLoadForm**, **BeforeLoadData**, **BeforeShowForm**, **BeforeDataSubmit**, **BeforeCloseForm** and **OnEvent**). Form events are out of scope at this time. Comment and Extension elements are covered in The EBT Comment Element and The EBT Extension Element.

4.3.2 Introduction to the XFCs

We will *briefly* survey the XFCs in this section in order to obtain a rapid overview, but we will return to them in detail in Introduction to SDC Basic Schema Types and The XFCs and their DEF Representations.

```
<Section ID="S1" title="I am a section" >
    <ChildItems>
        <DisplayedItem ID="DI1" title="I am a note on the  screen"/>

        <Question ID="Q1" title="This is a question title">
            <ListField>
                <List>
                    <ListItem ID="LI1" title="ListItem.title.1"/>
                    <ListItem ID="LI2" title="ListItem.title.2"/>
                    <ListItem ID="LI3" title="ListItem.title.3"/>
                </List>
            </ListField>
        </Question>
    </ChildItems>
</Section>
```

Example 4: XFC Nesting with a ChildItems Element: Section, DisplayedItem, Question and `ListItem`

Example 4 shows a simple SDC XML snippet representing the four most common XFCs: **`Section`**, **`DisplayedItem`**, **`Question`** and **`ListItem`**, along with a minimum number of SDC attributes. This is one of the simplest SDC DE structures in an FDF: A **`Section`** that wraps a single **`DisplayedItem`** and a **`Question`**. For now, note that the following features:

- A **`DisplayedItem`** and a **`Question`** are subsumed by a **`ChildItems`** element.
- The **`Question`** structure subsumes a **`ListField`**/**`List`** element structure before the **`ListItem`** elements appear.
- Each XFC has an **`ID`** and a **`title`** attribute. The **`ID`** is required in the XML, but the **`title`** is optional.

4.3.2.1 XFC Nesting

The **`ChildItems`** element, such as the one in Example 4, provides a hierarchical nesting wrapper for child S, Q, DI, BA and IF XFCs; S, Q and LI may subsume a **`ChildItems`** element and become parent XFCs. A parent XFC must use a **`ChildItems`** element to wrap child XFCs. The **`ChildItems`** wrapper element serves to separate XFC descendants from other parent-owned metadata elements (such as **`Property`**, **`Comment`** and **`Event`** elements) that supplement the parent XFC. S, Q and LI may be nested to any depth.

Question does not use the **`ChildItems`** wrapper for containing **`ListItem`**s. Instead, **`ListItem`**s appear in a **`/Question/ListField/List/ListItem`** XML construct without a **`ChildItems`** wrapper. This exception exists because the **`ListField/List`** structure serves as a wrapper for all **`ListItem`**s belonging to a **`Question`** element.

Any XFC, except for LI, may be nested under a **`ChildItems`** element. Thus, a child **`Section`** or **`Question`** or **`ListItem`** can also wrap one or more XFCs if they are contained in a **`ChildItems`** element.

The general rules for XFC nesting are:

- The following XFCs may have descendant XFCs S, Q and LI. The XFC descendants are always wrapped inside a **`ChildItems`** parent wrapper.
    - DI, BA, and IF do not support descendant XFCs. However, IF may inject SDC XML that contain descendants (see The InjectForm XFC (draft)).
- The following XFCs must have a direct **`ChildItems`** parent: S, Q, DI, BA, IF.

- The only exceptions in the above list are the **Header**, **Body** and **Footer** sections that exist under the **FormDesign** element.

- **LI** is an exception to the other XFCs. An LI always appears in a **/Question/ListField/List/ListItem** construct.

4.3.2.2 The XFC Building Blocks

Please note the following basic concepts about XFC common features. We will cover the XFCs individually in detail in Introduction to the XFCs.

- Only **Question** and **ListItem** are involved directly in DEF data capture.
- **Section** and **DisplayedItem** are used to support display, styling, and organization of the DEF.
- Only **Section**, **Question** and **ListItem** can subsume nested child XFCs using the **ChildItems** element.
- **ListItem** has a close structural relationship with **Question**.
- SDC supports user-controlled repeating of **Section** and **Question** XFCs, which allow a user to enter data for the repeating **Question**s multiple times, when appropriate. Any SDC XML that is subsumed by the repeating XFC elements is repeated with them.

We will now introduce just enough information about each XFC element to get a broad understanding of the basic FDF structure. We will return to each XFC in detail later.

Section (S): The most basic structure of an FDF is the **Section** XFC. As noted above, three specially-named sections are the **Header**, **Body** and **Footer**, which appear as direct children of the top-level **FormDesign** element.[24] No other XFCs are allowed directly under the **FormDesign** root element. The main purpose of the **Section** XFC is to group other XFCs (including other **Section**s) into blocks of content that make sense for users of a DEF. **Section**, along with any subsumed XML, may be repeated inside the DEF.

DisplayedItem (DI):

The **DI** element is used to define visible text[25] for display almost anywhere in a DEF. Each **DI** has a unique **ID**, but, unlike **Section** and **Question**, a DI cannot have XFC descendants. Thus it cannot be repeated unless it is subsumed inside a repeating XFC (S or Q). Displayed text that requires the use of XFC descendants must use the **Section** element instead. The formatting of any SDC element, such as DI or **Section**, can be specified by the **styleClass**[26] attribute.

Question (Q):

Questions may appear in two basic forms.

- **Question**s that can capture a response (fill-in) value directly are called **Question-Response (QR)** items**.** They are sometimes also called Question Fill-in (QF) items. **Response** metadata and captured **Response** values are handled by the **ResponseField/Response** XML element structure.

[24] The attributes directly under **FormDesign** are described later.

[25] A DI, like all XFCs, may also display rich content such as images, links or video.

[26] **styleClass** is available on nearly all SDC elements. It is intended to assist with formatting if the element is displayed in a DEF.

- Other **Question**s take a list of answer choices, called **ListItems** (**LI**). In the most common cases, **Question**s with **ListItem** choices are either **single-select** (**QS**) or **multi-select** (**QM**). The **Question** element defining QS or QM can never have a **ResponseField/Response** element under the **Question**.

ListItem (LI):

ListItems come in two basic types:[27]

- The simple **ListItem** is sometimes called an "answer choice" or "pick list item."
- A **ListItem** that, when selected by the user in the DEF, can capture the user's response, is called a **ListItem-Response** item (**LIR**), but may also be referred to as an "answer fill-in" (**AF**). An LIR is a LI that subsumes a **ListItemResponseField** element structure in the FDF XML. The LIR structure is **ListItem**/**ListItemResponseField/Response**. We will cover the LIR pattern later.
- **ListItem**s are always wrapped together as a group, nested inside a **Question/ListField/List** element structure.[28]

ButtonAction (BA):

All of the XFCs can trigger actions in the DEF that affect the appearance, behavior and data responses in the DEF. However, **ButtonAction** provides a user-controlled visible screen area (a "button" control) to trigger any kind of action in a DEF. **ButtonAction** translates into a visible screen area that can be clicked to trigger an arbitrary action in the DEF. ButtonAction adds one new element to its DisplayedType ancestor: The **OnClick** event. This event is fired whenever the user clicks on the displayed screen area represented by **ButtonAction**.

Controls that implement ButtonAction need not assume the physical layout of an onscreen button. They may use the **Link** and **BlobContent** features to customize the DEF appearance.

InjectForm (IF):

InjectForm is a kind of "virtual XFC" because it acts as a placeholder where XFCs from the same FDF or other FDFs may be injected into any location in the current FDF. At the top level of **InjectForm**, a single **Section** or **Question** or an entire FDF under a single **FormDesign** element may be injected inside the **InjectForm** element. The injected parts may be FDF-derived, or FDF-R-derived, such that they can contain previously-entered historical instance data. FDF parts to inject under **InjectForm** are identified by the SDC package that contains the form, and package identifiers rather than FDF identifiers are therefore used for this purpose

4.4 FormDesign Attributes and Properties

The FDF root element **FormDesign**[29] has a relatively large number of attributes. Some of the attributes address ***static*** FDF metadata inserted by the FDF designer. These are related to the identity of the empty FDF (i.e., before any data are entered into the FDF).

Other attributes have values that are used to track form ***instance*** data. These instance attributes are added to the **FormDesign** element to track individual instances of forms that are instantiated for a specific purpose, e.g., entering data on a specific patient during a single encounter. Instance values are assigned by the FM and/or FF.

For tracking purposes in some use cases, a FM may assign some of the values for the instance attributes before delivering an SDC form to the FF, and the FF must check for these pre-assigned instance-tracking values to avoid

[27] **ListItem**s can be handled as single-select or multi-select. This will be discussed in The Question XFC.

[28] Other uses for **ListField** and its substructure will be explained in The ListField Element.

[29] An FDF root element may be either **FormDesign** or **DemogFormDesign**. The use of the latter element will be explained when we cover the SDC Package structure. For now, we focus on **FormDesign**, but the discussion applies identically to the **DemogFormDesign** as well. **DemogFormDesign** is identical in every way to **FormDesign**, except for the name of the topmost (root) element.

inappropriately overriding FM-assigned values. In other cases, the instance tracking values are assigned by the FF, and included as part of the forms data set that is transmitted to the FR. If the FF detects that the FM has not assigned instance values to the instance attributes, then it may assign them itself. A FR should not alter the content of a received FF form (i.e., the FDF-R). However, a FR may log some or all of the instance attribute values in another data storage area, and may add its own logging information (e.g., time of receipt, hash value of XML contents, etc.), but these data should not be added to the FDF-R.

In many cases, the FF is a "dumb" web page, and therefore the FF assigns none of the values for the instance attributes. In some cases, the FM and/or FF and/or FR may be part of the same institutional system or collaborating network, and this may allow standardized role assignment to the FM and FF actors, which will determine which actor(s) assign the instance values.

All instance attributes are optional in the XML Schema, because they are not relevant in empty (non-instance) forms. Some static attributes are also optional. A FR should check for the correct assignment of all attribute values as part of the validation process for a received form. The FR may need to reject SDC data that does not include valid values for **FormDesign** attributes. The reason for rejection should be reported back to the FF for display to the FF user, so that the user may be able to correct the invalid data before resubmitting.

A third class of attributes are the ***instance status*** attributes. These are assigned by the FF based on information provided by the DEF user. These attributes address the type of changes in the DEF and the finality of the submitted FDF-R.

All required attributes are marked with `required`.

```
<FormDesign

Namespaces
  xmlns="urn:ihe:qrph:sdc:2016" required
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:h=http://www.w3.org/1999/xhtml

Schema-based validation
  xsi:schemaLocation="urn:ihe:qrph:sdc:2016 file:SDCFormDesign.xsd"

generic BaseType attributes
  order="0"
  type="Bmk"
  styleClass=""
  name="CAPeCC_Lung.Bmk"

identifier-related static attributes
  baseURI="cap.org" required
  lineage="Lung.Bmk.227" required
  version="1.001.011.RC1" required
  versionPrev
  ID="Lung.Bmk.227_1.001.011.RC1_sdcFDF" required
  fullURI="_baseURI=cap.org&amp;_lineage=Lung.Bmk.227&amp;_version=1.001.011.RC1&amp;
  _docType=sdcFDF" required
  prevVersionID="_baseURI=cap.org&amp;_lineage=Lung.Bmk.227&amp;
  _version=1.000.001.REL&amp;_docType=sdcFDF"
  basedOnURI=""

other FDF attributes
  filename="Lung.Bmk.227_1.001.011.RC1_sdcFDF.xml"
  formTitle="Lung Biomarker Reporting Template"

data-submission (instance) identifiers for FDF-R
  instanceID="" required in FDF-R
  instanceVersion="" required in FDF-R
```

```
  instanceVersionPrev=""
  instanceVersionURI=""

Instance Status Attributes for FDF-R
  approvalStatus=""
  completionStatus=""
  newData=""
  changedData=""

FormDesign Property elements
  <Property name="Copyright" type="CAPeCC_static_text" styleClass="copyright"
    order="1" propName="Copyright"
    val="(c) 2018 College of American Pathologists.  License required for use."/>
  <Property name="ApprovalStatus" type="CAPeCC_meta" order="13"
    propName="ApprovalStatus" val="RC1"/>

Main section of form starts here
  <Body name="Body" order="14" ID="Lung.Bmk.227_1.001.011.RC1_sdcFDF_Body"/>
</FormDesign>
```

Example 5: FormDesign attributes and Properties

4.4.1 FDF Namespaces and the Namespaced Attributes

- `xmlns`**= "urn:ihe:qrph:sdc:2016"** indicates the default SDC namespace. No namespace prefix is needed.
- `xmlns:xsi` - This is the namespace declaration that allows the inclusion of XML Schema instance attributes in the SDC XML. The use of the "xsi" namespace prefix is only used to allow the inclusion of the `xsi:schemaLocation` attribute in the SDC XML.
- `xsi:schemaLocation` attribute has two values, separated by a space. The first value is the SDC namespace. The second value is the location of the XML Schema to use with that namespace.[30]
- `xmlns:h=`http://www.w3.org/1999/xhtml is a namespace that allows us to include HTML rich text in SDC forms. This namespace declaration may be located alternatively inside each SDC **`HTML`** element.

4.4.2 FormDesign Attributes

`FormDesign` Static Attributes: Values for these attributes are supplied by the form designer, and do not change in instance documents after data capture.

- `formTitle`**:** [O] [def: ""] [dt: string] Human readable title for display when choosing forms from list provided by a FM. The `formTitle` may be displayed at the top of the DEF.
- `baseURI`: [O] [def: ""] [dt: anyURI] This parameter is required in the **`FormDesign`** element but is optional in the XFCs. It identifies the organization and/or group that is responsible for designing and maintaining the FDF or XFC. It's best to avoid using prefixes like "http://" or "https://" in this attribute because these can occasionally cause XML validation errors when used in a URI-typed field. Example: `"myorg.net/myGroup/2019"`
- `basedOnURI`**:** [O] [def: ""] [dt: anyURI] URI used to identify the SDC form that that the current FDF is based upon. In most cases, this should be a standard SDC form that is modified and/or extended by the current FDF. It's best to avoid using prefixes like "http://" or "https://" because these can occasionally cause XML validation errors when used in a URI-typed field. The URI format should be the same format used in `fullURI`, which is patterned after the SDC web service API.
- `lineage`: [dt: string] A string identifier that is used to group multiple versions of a single form. The lineage is constant for all versions of a single kind of form. When appended to `baseURI`, it can be used to retrieve all versions of one particular form. Example: `lineage="Lung.Bmk.227"`

[30] All SDC Schemas are available at https://github.com/IHE-SDC-WG

- **version**: [dt: string] A string that contains the version text for the current form. It is designed to be used in conjunction with **baseURI** and **lineage**.
- **versionPrev:** [O] [def: ""] [dt: string] Identify the immediate previous version of the current FDF. The format is the same as **version**. The primary role of this optional attribute is to allow automated comparisons between a current FDF and the immediate previous FDF version. This is often helpful when deciding whether to adopt a newer version of an FDF.
- **ID**: [dt: anyURI] In **FormDesign**, the **ID** is used to uniquely identify an "empty" FDF (i.e., *not* an FDF-R, which contains user-entered data) from other FDF files that have different FDF content. Thus, a **FormDesign ID** is shared by all FDFs that have the same **lineage** and **version**. To represent this concept, we create the **ID** content with the following formula:
  lineage_version_sdcFDF.
  In other words, we concatenate the components **lineage**, **version** and the text "sdcFDF" using "_" characters as separators. The suffix "sdcFDF" indicates that we are working with the FDF variant, not the FDF-R or the DE variant of SDC XML. From Example 5:

  ```
  ID="Lung.Bmk.227_1.001.011.RC1_sdcFDF",
  ```

  we can see how **lineage** and **version** values and the text "sdcFDF" are concatenated with "_".
- **filename:** [O] [def: ""] [dt: string] The filename of the FDF when is saved to a file storage device (e.g., a disk or USB drive). The **filename** appears inside the FDF XML to help ensure the identity of the FDF content in case the saved filename (on a disk drive, etc.) has been changed for any reason.
  - For FDF files without response data, one suggested format for **filename** is:
    - **lineage_version_**sdcFDF.xml.
  - For assigning a **filename** to FDF-R instance documents, the following **filename** pattern may be used:
    - **lineage_version_instanceID_instanceVersion_**sdcFDF**R**.xml.

      instanceID is a GUID that identifies an instance of an FDF-R document. **instanceVersion** is the version of the instance FDF-R. However, this format can become rather long, and thus short GUIDs or other formats may be considered.
    - An alternative shorter format for an FDF-R **filename** is:
      - **instanceID_instanceVersion_**sdcFDF**R**.xml

        However, this format is less human-readable than the longer format that includes the **lineage** and **version** information.
  - FDF-R files should not reuse the **filename** of the empty FDF. It is better to delete the attribute in the FDF-R than to reuse the FDF's **filename** value.
  - The **filename** datatype is string. It is not required.
- **fullURI**: [dt: anyURI] The full URI that uniquely identifies the current form. The URI is patterned after the SDC web service API. It is created by building up a REST-style query string from several components: **baseURI**, **lineage**, **version** , and **doctype**, using a format of component=value pairs, as in the following example:

  ```
  fullURI="_baseURI=cap.org&amp;_lineage=Lung.Bmk.227&amp;_version=1.001.011.RC1&amp;docType=sdcFDF".
  ```

Note that each bolded component name is preceded by an underscore "**_**" and that component names and values derive from a **FormDesign** attribute. Note that all "**&**" (ampersand) symbols are escaped using the standard XML/HTML "&" escape notation. The specific order of components shown in the URI is not required, but the displayed order is suggested for consistency and readability.

The only **fullURI** component not present as an attribute in **FormDesign** is **docType**, which is always set to "sdcFDF" for FDFs, and is set to "sdcFDFR" for FDF-R documents, which contain data. For other SDC document types, **doctype** is set to "sdcDE" for DEs (including CDEs in registries), "sdcPkg" for SDC Packages, and "sdcMap" for SDC maps.

Note that the FM endpoint URI is not present in **fullURI**. This information may be found in the SDC Package, under the **SDCPackage/Admin/RegistryData** element. It may also be provided in a custom FDF **Property** if desired.

FormDesign Instance Attributes: These are attributes found in an FDF-R, which is created to contain user-entered data. Included in brackets "[ ]" are the IHE actor(s) that are responsible for adding the correct value to each attribute. Note that the **instanceVersion** attribute is a Timestamp, and thus must be assigned by the FF.[31]

- **instanceID:** [FM or FF] [O] [def: ""] [dt: string] Unique string (e.g., a GUID) used to identify a unique instance of a form, such as a form used during a single patient encounter. The **instanceID** is used to track saved form responses across time and across multiple episodes of editing by end-users. This string does not change for each edit session of a form or package instance. Datatype = string. The **instanceID** is required in an FDF-R; It is not allowed in an FDF.
- **instanceVersion**: [FF] [O] [def: ""] [dt: dateTime] Timestamp used to identify a unique instance of a form. Used for tracking form responses across time and across multiple episodes of editing by end-users. This field must change for each edit session of a form instance. Datatype = dateTime. The **instanceVersion** is required in an FDF-R; It is not allowed in an FDF.
- **instanceVersionURI:** [FF] [O] [def: ""] [dt: anyURI] Globally-unique URI used to identify a unique instance of a Package with saved FDF-R responses from the current FDF-R. It is used for tracking FDF-R and Package responses across time and across multiple episodes of editing by end-users. The **instanceVersionURI** must change for each edit/save session of a form instance (defined by **instanceVersion**).

  The **instanceVersionURI** should be formatted similarly to the **fullURI** but must include values for **instanceID** and **instanceVersion**. The **instanceVersion** value is the release date/time for the new version, in W3C datetime format. An example **instanceVersionURI** is:

  - **instanceVersionURI**="**_baseURI**=cap.org&**_lineage**=Lung.Bmk.227&**_version**=1.001.011.RC1 &**_instanceID**=Abc1dee2fg987&**_instanceVersion**=2019-07-16T19:20:30+01:00&**_docType**=sdcFDFR "

  It is possible to create a shorter URI without the **_baseURI, _lineage** and **_version** parameters, as long as the URI is able to globally and uniquely identify and retrieve the instance and version of the FDF-R that was transmitted:

  - **instanceVersionURI**="**_instanceID**=Abc1dee2fg987&**_instanceVersion**=2019-07-16T19:20:30+01:00&**_docType**=sdcFDFR"

[31] The FM is generally not involved with the DEF after it sends the FDF to the FF, and thus cannot write the required timestamps into the FDF-R. Therefore, any attribute value requiring a timestamp must be added by the FF. Since the FR is not allowed to alter the content of a submitted FDF-R, it cannot add timestamps to the FDF-R either.

Note that the FR webservice endpoint URI is not provided in the **instanceVersionURI**. The FR endpoint and its security settings may be found in the SDC Package that contains the FDF-R, at **SDCPackage/SubmissionRule**. An FR may also be provided in a custom FDF **Property** if desired.

The **docType** value for **instanceVersionURI** is sdcFDFR for a single FDF-R transaction; sdcFDFR should also be used when the Package contains an optional **DemogForm** FDF-R in addition to a single **FormDesign** FDF-R as content. The docType value for a Package with multiple FDF-R components and/or other content is sdcPkg. The specific order of components shown in the URI examples is not required, but the component order shown above is suggested for consistency and readability.

The **instanceVersionURI** is not required in an FDF-R, and is not allowed in an FDF.

- **instanceVersionPrev**: [FM or FF] [O] [def: ""] [dt: dateTime] Timestamp value to identify the immediate previous instance of an FDF-R instance. Used for tracking FDF-R responses across time and across multiple episodes of editing by end-users. This field must change for each edit session of an FDF-R instance. The **instanceVersionPrev** is not required in an FDF-R or an SDCPackage, and may only appear in an FDF-R; it is not allowed in an FDF.

FormDesign Instance Status Attributes: The following instance attributes are generally assigned by the FF, based on input from the DEF user:

- **approvalStatus**: [FF] [O] [def: ""] [dt: string] Describes report fitness for clinical or other action: **inProcess**: currently being edited, users should not rely on results; **preliminary**: report is awaiting final review and approval; **approved**: report is fit for clinical or other action; often synonymous with final; **cancelled**: report/procedure has been aborted before being issued; **retracted**: report has been deemed unfit for clinical or other action. Not required in an FDF-R, and not allowed in an FDF.
- **completionStatus**: [FF] [O] [def: ""] [dt: string] The extent to which a report contains all of the requested information: **pending**: no information is yet available; **incomplete**: some requested information is not yet available; **complete**: all information is available in the requested report. Not required in an FDF-R, and not allowed in an FDF.
- **newData**: [FF] [O] [def: null] [dt: boolean] Identifies existence of data that is new to the current instance of package, form, section, or question compared to the previous instance of the package, form, section, or question. Not required in an FDF-R, and not allowed in an FDF.
- **changedData**: [FF] [O] [def: null] [dt: boolean] Identifies existence of data that has been changed in the current instance of package/form/section/question compared to the previous instance of the package/form/section/question. Not required in an FDF-R, and not allowed in an FDF.

4.4.3 FDF and XFC Identifiers (IDs)

Each FDF has its own **ID** attribute in the **FormDesign** element, which uniquely identifies each "class" of FDF document. The word "class" is meant to convey that the **lineage** and **version** are identical for each FDF that shares an FDF **ID**; In FDFs that share an **ID**, the XML structure and metadata are identical.

In addition, every XFC inside an FDF contains **ID** content, which must be unique within its FDF. However, XFC **ID** content values do not need to be **globally** unique. Thus, the same XFC **ID** values may be used in other FDFs, including FDFs with the same **lineage** (but different **version**s) and FDFs from a different author or source. Within a given organization (defined by the **baseURI**), it is important to tightly control the release of **ID**s, so that the same **ID** never appears in a different FDF **lineage**. Controlling the release of XFC **ID**s to be unique across lineages has major advantages. This approach reuses an XFC **ID** to be only when that XFC is copied unchanged into a new FDF **version** derived from the same FDF **lineage**. This approach allows **ID**-based queries across multiple FDF **version**s that share a common **lineage**.

Do IDs change when a new version of an FDF is released? Most XFC **ID**s are not changed when an FDF from a given **lineage** is released with a new **version**. However, a new FDF **version** will force a change for the **ID** on

the **FormDesign** element. Within a given lineage of FDFs, identical XFC ID values signal that the semantics and basic structure of the object remains unchanged across different versions. The criteria for changing IDs on versioned XFC items are discussed later in this document.

It is possible to create a system for globally unique ID values that would enable every FDF and every XFC in a form to be uniquely identified, e.g., by using GUIDs. However, the use and maintenance of long GUID ID content can sometimes cause difficulties and errors.

Do IDs ever repeat? Within a single baseURI organization, a given **XFC** ID is found only in one lineage of FDFs. Similarly, within a single baseURI organization, a given **FDF** ID is found only in a single version of a single lineage of an FDF class. All IDs in an FDF must be unique within the FDF. It is worth nothing that FDF IDs and XFC IDs generally use very different formats, so the chance of a ID collision between an FDF and its contained XFCs is nearly zero. Once an ID is released, it cannot be reused. If an ID is deprecated, it may not be reused.

Why aren't IDs always in sequential order? IDs are derived from a pre-existing list of sequential integers that represent unused IDs. The unused IDs are available to all FDFs that are being edited. Sometimes during the initial phases of FDF editing, blocks of temporary XFCs are created, only to be removed before release. The unused IDs from the deleted records may be reused, resulting in the reuse of lower-numbered IDs that appear out of sequence. These lower-numbered IDs are still unique, but no longer in a strictly-increasing sequence.

In other cases, XFCs (or entire blocks of XFCs) may be moved, during the FDF modeling process, to different locations within the same FDF. (IDs are never moved to other FDFs.) Because lower-numbered IDs may be moved below higher numbered IDs in an FDF, they are no longer in a numerically-increasing sequential order.

If the IDs can change between versions, how can we combine data from multiple versions? Before attempting cross-version queries in a single FDF lineage, it is important to ensure that all **ListItem**s for the **Question** being queried are essentially unchanged across all the FDF versions. Blindly querying across different versions of a **Question**, when the different versions have different **ListItem** sets, can return incorrect results in some cases.

When **ListItem**s for a **Question** change, the **Question** (and its ID) is deprecated and replaced. If the data analyst is aware of the change implications, it should be possible to combine similar FDF-DE (**Question-ListItem**) blocks into a common query and result dataset. In these cases, interpretation of the query results requires that the changes over time (i.e., in different FDF versions) be considered.

4.4.4 The baseURI Namespace

To enable the use of simpler, shorter ID values that are easier to use, the SDC Schema includes the baseURI attribute. The baseURI functions similarly to an XML namespace since it uniquely identifies the organization and/or group that authored and maintains the FDF. When the baseURI value is used in conjunction with the ID value, a composite globally-unique identifier (**CGUI**) is created for each XFC. For example, if the simple ID value is "100", and the baseURI is "cap.org/FormDomainSDC1/", then the CGUI is " cap.org/FormDomainSDC1/100", which the issuing organization (cap.org) should guarantee as unique within an FDF lineage. The CGUI is generated as needed and thus is not found explicitly in the FDF XML. Note that the ID value "100" need not be globally unique, but the CGUI should be globally unique.

The default XFC baseURI is defined as identical to the **FormDesign** baseURI. A new baseURI is not assigned to an XFC unless the **FormDesign** baseURI is not appropriate for a specific XFC and its descendant XFCs. An appropriate baseURI should reflect the organization that created and/or maintains the XFC content, as well as the FDF content domain that it addresses. The baseURI is inherited by all descendant XFCs and should not be added to the SDC XML unless there is a change in baseURI. In most cases, XFCs do not receive a new baseURI. A new baseURI is generally assigned only when importing copyrighted material, or in other situations where the XFC content is maintained separately from the main FDF content. For example, injection of content from another FDF may require the use of a new baseURI.

Ideally, the baseURI is assigned once at the highest level of an FDF (the **FormDesign** element). Every descendant ID then inherits the baseURI, without repeating the baseURI at each XFC. If necessary, multiple baseURIs may be used throughout an FDF; these override any higher-level URI assigned above it in the FDF hierarchy. Thus, an

XFC uses the `baseURI` ancestor that is closest to it in the FDF hierarchy, starting with its own `baseURI` value (if it is present).

To create a `baseURI`, the institution that creates an SDC form should have one or more registered globally-unique IDs (e.g., domain names or GUIDs) that are used to uniquely[32] identify the origin and uniqueness of its SDC forms. Ideally, these `ID`s should be in URL format, and should ideally represent real URLs that can be "dereferenced" to provide information about the organization and its forms.[33] It's best to avoid using prefixes like "http://" or "https://" in the `baseURI` text because these add unnecessary length and can occasionally cause XML validation errors when used in a URI-typed field.

[32] The uniqueness of the `baseURI` value can be guaranteed by using a namespace registry such as the Internet DNS system, administered by ICANN and IANA.

[33] In the example above, the dereferenced CGUI, https:// www.cap.org/FormsDomainSDC1/100, could be typed into a web browser to obtain information about that XFC, and possibly information about the specific FDF that contains it, and the group or author that created it. However, the ability to dereference a baseURI is determined by the use case.

4.4.5 FormDesign Properties (eCC)

Earlier, we introduced the concept of **Property** elements under the **FormDesign** tag. FDF **Properties** allow form designers to introduce domain-specific metadata into the FDF for a variety of purposes. Any of these **Property** elements may be displayed (or not) in the FDF, depending on the use case, and under the control of the FF software.

To provide some examples, we list the CAP eCC **Property** types for the **FormDesign** element:

```
<Property name="Copyright" type="CAPeCC_static_text" styleClass="copyright"
propName="Copyright" val="(c) 2018 College of American Pathologists.  All rights reserved.
License required for use." />
<Property name="GenericHeaderText" type="CAPeCC_static_text" propName="GenericHeaderText"
val="Surgical Pathology Cancer Case Summary (Checklist)" />
<Property name="Category" type="CAPeCC_meta" propName="Category" val="Endocrine" />
<Property name="OfficialName" type="CAPeCC_meta" propName="OfficialName" val="ADRENAL GLAND" />
<Property name="CAP_ProtocolName" type="CAPeCC_meta" propName="CAP_ProtocolName" val="Adrenal
Gland" />
<Property name="CAP_ProtocolShortName" type="CAPeCC_meta" propName="CAP_ProtocolShortName"
val="Adrenal" />
<Property name="CAP_ProtocolVersion" type="CAPeCC_meta" propName="CAP_ProtocolVersion"
val="4.0.1.1" />
<Property name="TemplateID" type="CAPeCC_meta" propName="TemplateID" val="129.100004300" />
<Property name="Restrictions" type="CAPeCC_meta" propName="Restrictions" val="Please refer to the
cancer protocol cover page (www.cap.org/cancerprotocols) for information about which tumor types
and procedures can be reported using this template." />
<Property name="CAP_Required" type="CAPeCC_meta" propName="CAP_Required" val="true" />
<Property name="AccreditationDate" type="CAPeCC_meta dt.dateTime" propName="AccreditationDate"
val="2/28/2018 12:00:00 AM" />
<Property name="WebPostingDate" type="CAPeCC_meta dt.dateTime" propName="WebPostingDate"
val="6/30/2017 12:00:00 AM" />
<Property name="ApprovalStatus" type="CAPeCC_meta" propName="ReleaseStatus" val="RC2" />
<Property name="AJCC_Version" type="CAPeCC_meta" propName="AJCC_Version" val="8th Edition" />
```

CAP **Property** types:

- `Copyright`: A copyright statement
- `GenericHeaderText`: Text that appears at the top of the DEF
- `Category`: The organ group that includes the current form, e.g., "Endocrine"
- `OfficialName`: The full human-readable name of the current form
- `CAP_ProtocolName`: The name of the CAP Cancer Protocol that contains the current form
- `CAP_ProtocolShortName`: The abbreviated name of the CAP Cancer Protocol that contains the current form
- `CAP_ProtocolVersion`: The version of the CAP Protocol
- `TemplateID`: A numeric identifier for the form **lineage**, appended to the **lineage** text
- `Restrictions`: Rules about when to use or not use this form
- `CAP_Required`: The value is "true" if the form is required for Commission on Cancer accreditation
- `AccreditationDate`: The data that this form must go into effect to satisfy the requirements of accreditation-related surveys
- `WebPostingDate`: The date the form was posted on the CAP website
- `ApprovalStatus`: A short text flag that indicates how close the form is to an officially-approved release. Examples include "**CTP1**" (Community Technology Preview) , "**RC2**" (Release Candidate) and "**REL**" (official Release for implementation)
- `AJCC_Version`: The version of the American Joint Committee on Cancer (AJCC) Staging Manual used in the FDF

5 Introduction to SDC Basic Schema Types

5.1 SDC Schema File Overview (TBD)

5.2 Schema files.

The basic SDC Schema that is used for designing FDFs is organized as a hierarchy of 5 files. Each file includes the files that precede it in the hierarchy. The lowest level is the SDCBase.xsd, and the top level is SDCFormDesign.xsd. The hierarchy is arranged as shown below, with the most inclusive files on top. Indented lines indicate that the indented Schema is *included* (i.e., using the `xs:include` Schema instruction) in the outdented Schema file above it.

SDCFormDesign.xsd - includes:
  SDCExpressions.xsd - includes:
    SDCResources.xsd - includes:
      SDCDataTypes.xsd - includes:
        SDCBase.xsd

Additional SDC Schemas are required for SDC retrieval transactions:

SDCRetrieveForm.xsd – includes:
  SDCFormDesign.xsd
  SDCTemplateAdmin.xsd
  SDCMappings.xsd

And one Schema used for FDF and FDF-R transmissions to FRs

SDCSubmitForm – includes:
  SDCFormDesign.xsd

At this point, we will be concerned only with **SDCFormDesign** and its sub-Schemas.

5.2.1 Basic SDC Schema Type Hierarchy

To best understand the attributes and features supported by the various XFCs, we need to consider the SDC Schema inheritance model. Note that the word "Schema" is capitalized, which, in this document, indicates that we are dealing with an official W3C version 1.0 XML Schema to define the SDC architecture.

We now need to look at the layered Schema Types from which the XFCs derive. In the SDC Schema architecture, all XML elements are backed by an XML Schema Type to better support automated code generation from the Schema. This document will often use the capitalized word "Type" to indicate that we are discussing the element's definition as a Schema Type, and not the SDC XML element that derives from it and which is found in the FDF. In SDC, all Types and elements (except for W3C-derived datatypes) are capitalized, and all attributes use camelCase.[34]

The hierarchical arrangement of the SDC Schema Types within the Schema files is essential to understanding inherited attributes and elements. It is important to understand that the hierarchical arrangement of Schema files is not the same relationship as the hierarchical arrangement of Types within the and between the Schema files. SDC Schema Types are derived by inheritance from and aggregation of more primitive Types in the various included files in the Schema hierarchy.

Note that *abstract* Types are Schema Types that are never directly used to create XML elements in any SDC XML file. However, new Schema Types may be *derived* from an *abstract* Type, and these derived Types may be used to define an XML element.

[34] camelCase is a format that starts with a lower-case letter and uses capitals at the start of new words. Spaces are not permitted.

SDC *datatypes* include almost all W3C datatypes, and also include special SDC types that subsume HTML and XML blocks inside an FDF. These specialized HTML and XML blocks obey other Schema definitions, and thus must have their own XML namespaces to be included in SDC XML documents. HTML blocks must use the xhtml XML Schema, which is included with the SDC Schema set.

For consistency with W3C naming conventions, W3C datatype element names begin with *lower* case letters, unlike all of the other native SDC elements, which begin with *upper* case letters.

We now introduce the SDC Schema Types. In the following SDC Schema Type hierarchy (Example 6), the most primitive types are on top, and the most derived types are on the bottom. We present the Types in this manner to mirror a common approach to represent object trees in Object-Oriented Programming (OOP) languages, where each child object type points upwards to its "parent" (i.e., more primitive) type.

We follow this convention to highlight the deep connection between the SDC Schema Type definitions and the programming objects that may be created automatically from each Type.

BaseType (abstract)

(all SDC Datatypes and derivatives)
CodeMatchType
CommentType
ExtensionType
ExtensionBaseType (abstract)

(all Event, Rule, Action types and subtypes)
(all items defined in the SDCRetrieveForm sub-Schema)
PropertyType
TemplateAdminType and most included Types
ChildItemsType
LinkType, BlobType, ContactType and most included Types
CodedValueType, CodeSystemType
ListFieldType, ListType, LookupEndpointType
ResponseFieldType, ListItemResponseFieldType
BasePackageType
RetrieveFormPackageType[35] (**SDCPackage**)
IdentifiedExtensionType (abstract)

DataElementType (**DataElement**)
FormDesignType (**FormDesign**, **DemogFormDesign**)
InjectFormType (**InjectForm**)
DisplayedType (**DisplayedItem**)

ButtonItemType (**ButtonAction**)
ListItemType (**ListItem**)
RepeatingType (abstract)

SectionItemType (**Section**)
QuestionItemType (**Question**)

Example 6: SDC inheritance model

[35] Green font indicates a top level SDC Schema type. These SDC Types define elements that are at the top level of the various SDC XML documents, such as **SDCPackage**, **FormDesign**, **DemogFormDesign** and **DataElement**.

To avoid confusion with the explanations used for Type trees, we need to review some terminology. The Types towards the top of the hierarchy are described with terms such as "parent," "base," "primitive," "supertype" and "low-level." Some terms like "low-level" and "base" can be confusing to some people because the low-level (base) types are at the top of the tree. These terms are used because they are the low-level building blocks from which more "higher-level" complex Types are derived and assembled. The Types towards the bottom of the are thus described with terms like "high-level," "child," "subtype," "descendant," and "derived."

In some depictions of Type hierarchies, the top-bottom orientation of the tree is reversed, with the primitive types at the bottom, and the high-level (derived) types at the top. For example, a class diagram[36] for an object-oriented programming diagram will sometimes display the tree in the reverse orientation, with the most derived "top-level" types on top.

Four of the Types are marked with "*(abstract)*", which means that an XML element cannot be created from that Type; an XML element can only be created from a non-abstract *concrete* Type that inherits (derives) from the *abstract* Type, further down the hierarchy tree.

The XFC Schema Types,[37] which are used to create SDC XML elements, are shown in **bold**. XFC Type names are followed by parentheses containing the name of the element in the FDF XML, e.g., **ListItemType** (`ListItem`).

Five of the Types listed above are **parents** (shown in *italics*), i.e., they have 1st level descendants that inherit from them, and which pass their elements and attributes to all of their 2nd level descendants, and so on, through multiple levels of inheritance.

Finally, note that DataElementType and FormDesignType derive directly from *IdentifiedExtensionType (abstract)*. This derivation introduces a level of common ancestry between these two high-level Types and all the XFCs. The common ancestry is critical to the creation of a consistent and coherent SDC object model from the SDC Schema.

[36] For our purposes, the classes of an object-oriented languages are equivalent to the Types of an XML Schema.

[37] XFC Types are the SDC Schema Types that define the XFC elements in the FDF. Each Schema Type ends with the suffix “Type” and is otherwise similar in name to the XFC XML element that it defines.

5.3 The SDC Schema Inheritance Model and XFC Definitions

The following Example 7 shows a partial inheritance hierarchy "tree" of the main SDC Schema Types, derived from Example 6. As before, the Types at the top of the hierarchy are the most primitive and the elements and attributes defined by the Type are inherited by derived Types further down the hierarchy tree. Indented types inherit elements and attributes from the parent Type above it, and all parallel-aligned Types are siblings that inherit from the same parent Type.[38]

Note that all XFCs inherit from *BaseType (abstract)*, *ExtensionBaseType(abstract)*, and *IdentifiedExtensionType (abstract)*. All XFCs, except for **InjectFormType**, inherit from ***DisplayedType***. **SectionItemType** and **QuestionItemType** additionally inherit from *RepeatingType (abstract)*.

BaseType (abstract)

ExtensionBaseType (abstract)

IdentifiedExtensionType (abstract)

InjectFormType (`InjectForm`)

DisplayedType (`DisplayedItem`)

ButtonItemType (`ButtonAction`)

ListItemType (`ListItem`)

RepeatingType (abstract)

SectionItemType (`Section`)

QuestionItemType (`Question`)

Example 7: SDC Schema XFC Inheritance Hierarchy

The list below again enumerates the lower-level SDC Types (numbered 1-5). Underneath each Type is a bulleted line that lists `attributes` and `elements` introduced by each Type. Omitted from the Example are the five XFC Type descendants that have no descendants of their own (i.e., all XFC Types except ***DisplayedType***). All of the attributes and elements in the upper parts of the Type hierarchy structure are inherited by the derived types further down the list. Of all the inherited elements and attributes listed below in Example 8, only `ID` is required to be present in the FDF for every XFC in the FDF XML.

1) BaseType (abstract):
- `name`, `styleClass`, `type`, `order`

2) ExtensionBaseType (abstract):
- `Comment`, `Extension`, `Property`

3) IdentifiedExtensionType (abstract):
- `ID`, `baseURI`

4) DisplayedType (`DisplayedItem`):
- `Link`, `BlobContent`, `Contact`, `CodedValue`, `OnEnter`, `OnExit`, `OnEvent`, `ActivateIf`, `DeActivateIf`
- `title`, `enabled`, `visible`, `mustImplement`, `showInReport`

5) RepeatingType (abstract):
- `minCard`, `maxCard`, `repeat`, `instanceGUID`, `parentGUID`

Example 8: Inherited XFC Elements and Attributes

[38] For example, ButtonItemType, ListItemType and RepeatingType are siblings, in that each inherits from the parent Displayed Type.

To obtain the full list of inherited elements and attributes for *RepeatingType (abstract)*, we walk up the inheritance hierarchy to identify the ancestral elements and attributes. Example 9 shows the **elements** and attributes inherited by **Section**[39] and **Question**[40] elements, which are derived from *RepeatingType (abstract)* (as shown previously in Example 7):

- name, styleClass, type, order
- Comment, Extension, Property
- ID, baseURI
- Link, BlobContent, Contact, CodedValue, OnEnter, OnExit, OnEvent, ActivateIf, DeActivateIf
- title, enabled, visible, mustImplement, showInReport
- minCard, maxCard, repeat, instanceGUID, parentGUID

Example 9: Inherited Elements and Attributes for the XFCs that derive from RepeatingType

Section and **Question** add their own elements and attributes on top of those inherited from *RepeatingType (abstract).* This inheritance model makes it easier to document and implement properties of the various FDF elements, because only the elements and attributes introduced at each derived Type (e.g., QuestionType) need be documented, and the rest can be documented by reference to the inherited elements and attributes of the parent Type. This model is optimized for the automated generation of an object-oriented inheritance model from the XML Schema.

The following two lists show the new elements and attributes introduced by the XFC descendants of **DisplayedType** (Example 10) and **RepeatingType** (Example 11).[41] As before, the SDC XML element name for each XFC Type is shown in parentheses after the XFC Type.

ButtonItemType (**ButtonAction**):

Elements:

OnClick

InjectFormType (InjectForm):

Elements:

Section
Question
FormDesign

Attributes:

pkgID
pkgInstanceURI
pkgInstanceVersionURI
pkgBaseURI
pkgFullURI
pkgManagerUR
rootItemID

ListItemType (**ListItem**):

Elements:

ChildItems
ListItemResponseField

Attributes

Selected
selectionDisablesChildren

selectionActivatesItems
selectionSelectsListItems
selectionDeselectsSiblings
omitWhenSelected
associatedValue
associatedValueType

Response Reporting Attributes:

repeat
instanceGUID
parentGUID

Example 10: DisplayedType Descendants

[39] The XML element "**Section**" is defined by **SectionItemType** in the SDC Schema.

[40] The XML element "**Question**" is defined by **QuestionItemType** in the SDC Schema.

[41] **DisplayedType**, which defines the **DisplayedItem** element, is not listed again here because it is a parent type, and therefore its elements and attributes were listed in Example 8.

SectionItemType (`Section`):	**QuestionItemType** (`Question`):

Elements:
`ChildItems`
 	
Attributes:
`ordered`
 	
Response Reporting Attributes:
`repeat`
`instanceGUID`
`parentGUID`
	Elements:
	`ChildItems`
	
	Response Reporting Attributes:
	`repeat`
	`instanceGUID`
	`parentGUID`

Example 11: RepeatingType Descendants

The elements and attributes introduced by each XFC will be discussed in the sections below. It is important to remember that `Section` and `Question` inherit all elements and attributes from *RepeatingType (abstract)*, whereas `ListItem` and `ButtonAction` inherit from **DisplayedType**.

It is also important to note that only `Section`, `Question`, and `ListItem` can nest child XFCs directly inside a `ChildItems` element. `DisplayedItem`, `ButtonAction` and `InjectForm` do not support `ChildItems` XFC nesting, although `InjectForm` can support descendants by injecting XFCs directly under the InjectForm element.

We now address the elements and attributes introduced at each level of SDC Schema hierarchy

5.4 The BaseType (abstract)

The BaseType is an *abstract* Type that is used to define the most basic features of every element defined in the SDC Schema, including elements derived from W3C datatypes. All of the BaseType attributes are *optional* to use in SDC XML.

The BaseType attributes are explained below:

- `name`: [O] [def: ""] [dt: ID] A unique identifier of W3C type "ID,"[42] which must be unique within an XML document and must adhere to W3C NCName restrictions. It is similar to a unique name for a variable, control or object, used to provide the ability for programmatic manipulation of an element. It is typically assigned by the form designer or generated algorithmically. The value of `name` must be unique within an FDF and FDF-R, even when FDF sections are repeated in the XML. Unlike `ID`, `name` is not affected by the `baseURI`.

  To avoid problems when using `name` attributes with programming languages, form designers should restrict the `name` content to begin with a letter or an underscore and to only contain characters that are legal for variable names. These generally include letters, numbers and underscore. The use of other characters may result in errors, depending on the programming language employed. Since XML is case-sensitive, attribute values should not be made unique solely on the basis of alphabetic case, since many programming languages are case-insensitive and cannot distinguish names based only on case differences.

[42] The W3C **ID** type, derived from NCName, is not the same as the SDC `ID` attribute. The SDC `ID` is derived from the W3C URI type, which is much more permissive regarding the attribute's content format.

- **type**: [O] [def: ""] [dt: string] The **type** attribute may contain one or more custom metadata "tokens" for the element, chosen from a standardized list of terms. It uses the W3C NMTOKENS type, and thus supports multiple space-separated values.

  Tokens are short alphanumeric text strings, defined by the W3C Schema NMTOKEN specification, that are defined in a use-case-specific Implementation Guide. The type NMTOKEN represents a single string token. NMTOKEN values may consist of letters, digits, periods (.), hyphens (-), underscores (_), and colons (:). They may start with any of these characters.[43] **type** tokens may be used to "decorate" one or more kinds of SDC elements. Multiple tokens in the **type** attribute are separated by whitespace.

  The list of terms (tokens) must be defined for each use case and are not defined explicitly by the SDC Schema model or the IHE SDC Profile.

  type tokens may be used to specify special handling by an application, and are usually used to define form display constraints, but may include other custom metadata as well. Style metadata should be handled with **styleClass** rather than **type**. Some **type** token examples include `tooltip`, `statusLineText`, etc. In general, **type** metadata should not affect the information content of a form.

- **styleClass**: [O] [def: ""] [dt: string] Holds developer-assigned class names for display styling, generally for use with an external style sheet. Like **type**, **styleClass** uses the W3C NMTOKENS XML type, and thus supports multiple space-separated values. Examples could be `alignTopLeft`, `thickSection`, `thinSection`, `align:bottom`, `pageBreak-after`, etc.

- **order**: [O] [def: null] [dt: decimal] A decimal attribute that allows the form template developer to define a sequential order for elements in a template. This serves the purpose of providing a definitive/original order to sections, questions, answer choices, etc., when required for display purposes. This is important when the original XML ordering may become disrupted due to the use of an implementation technology that does not natively support ordering (e.g., object collections), and it can also provide a check on the proper importing and ordering of the XML tree during implementation. The decimal datatype allows for insertion and repetition of new ordered SDC XML content into FDFs while preserving the original sequence of the XML without changing the original **order** content. For example, when inserting a new XML element between two existing SDC elements with **order** content of 1 and 2, the **order** content of the new element may be set to 1.5, so that the original elements need not have their **order** content changed.

[43] Source: www.datypic.com/sc/xsd/t-xsd_NMTOKEN.html

5.5 The ExtensionBaseType (abstract)

The ExtensionBaseType (EBT) is an *abstract* type that confers on descendants the ability to add custom **Extension**, **Comment** and **Property** elements. Most SDC Types inherit from ExtensionBaseType, with the notable exceptions of CommentType, ExtensionType, all SDC datatypes, and some direct derivatives of datatypes.[44]

5.5.1 The EBT **Property** Element

Property elements are based on PropertyType. PropertyType, which is a component of EBT, also derives recursively from EBT. As a consequence, a **Property** element may have any number of direct **Comment**, **Extension** and **Property** descendants, and the descendant **Property** elements may be nested recursively to any desired degree. **Property** elements may occur as children of all XFCs and most other FDF elements.[45]

Properties may be used to define standard or custom FDF metadata. They also may be used to record visible or hidden metadata for any purpose. **Property** elements, when used on a **DI** or descendants of DisplayedType, can be used for many kinds of tasks including:

- Display static information on forms
- Display context-sensitive information (e.g. tooltips, status bar text, help pop-ups, or special text designed for report output). (The **Link** element may also be used to provide help resources)
- Provide rich title text (i.e., in HTML format) for implementations that support rich text
- Provide alternative language text
- Provide ancillary, alternate, instructional or informational text for DisplayedType descendants

Each **Property** has a "name" which is found in the **propName** content. For example, if **propName** = "myProp", then we call it the "*myProp Property*". Each **Property** also has a "value" which is stored in the **val** attribute. When we refer to the "value" of a **Property**, we mean the content of the **val** attribute.

In most cases, **Property**-derived text is displayed only under certain conditions (e.g., for rendering tooltips or report output). Determining if/when a **Property** should be displayed on a DEF requires the interpretation of its **propName**, **type** and **styleClass** attributes, as specified by an established user community. Examples of possible **Property** **propName** content include "helpText", "tooltip", "statusBarText", "htmlTitle", etc., which are commonly supported concepts and control types in most DEF programming frameworks. The **Property** element is also commonly used to contain invisible form metadata, e.g., versioning, source references, alternate language text, etc. Any SDC **Property** value may optionally be available as strongly-typed data (e.g., integer, string, etc.). Common SDC **propName** content examples include "altText" and "reportText", which will be described later.

---

[44] Direct SDC derivatives of datatypes generally extend a datatype by adding one or more attributes. The derived datatypes are used to build up more complex Types, which *are* derivatives of ExtensionBaseType. This makes it possible to add **Property**, **Comment** and **Extension** elements to all derived SDC Schema Types and the elements defined by them.
It serves no purpose to be able to extend the ExtensionType.
The CommentType is intentionally kept simple and non-extensible. More complex commenting needs can be handled by the PropertyType.

[45] This refers to all SDC elements that descend from ExtensionBaseType.

Property may have descendants of **Property**, **Comment** and **Extension**. As shown in Example 12, **Properties** may be nested to any depth and use any datatype for the **Property**'s value.[46]

```
<DisplayedItem ID="DI1" title="?This is a Note">
    <Property propName="myPropName1" val="This is property value 1">
        <Property propName="myPropName1.1" val="This is property value 1.1">
            <Property propName="myPropName1.1.1" val="This is property value 1.1.1"/>
        </Property>
    </Property>

    <Property propName="myPropName2" val="This is property value 2">
        <Property propName="myPropName2.1" val="This is property value 2.1">
            <Property propName="myPropName2.1.1" val="This is property value 2.1.1"/>
        </Property>
    </Property>
</DisplayedItem>
```

Example 12: Nested Properties

Properties will be called by their **propName**. For example, if **propName** = "**myPropName1**", then the **Property** will be called the "**myPropName1 Property**." A **Property** must have content in the **propName** attribute. The value of the **Property** is stored in the **val** attribute. Strongly-typed **Property** value content is discussed next.

5.5.1.1 Strongly-Typed Property Values for Special Purposes

Sometimes, a **Property** value needs to use a specific datatype, and this datatype ideally should be validated in the SDC XML for the correct format required by that datatype. In these cases, we can use the **TypedValue** feature of the **Property** element. Using a **TypedValue** with a **Property** produces a "strongly-typed Property."

TypedValue datatypes include string, numeric, dateTime as well as XML and HTML. In the SDC Schema, **HTML** and **XML** are regular datatypes, in addition to the standard W3C datatypes, such as **string** and **integer**. However, **HTML** in the SDC Schema means the strict XHTML[47, 48] variant. SDC thus supports custom HTML and XML islands everywhere that **Property** and datatypes are supported in the FDF.

Example 13 shows a strongly typed **Property** used to include XHTML-formatted text at any point in any EBT-derived element. The term "strongly-typed" in this context means that we are using a well-known (and preferably well-documented) **propName** value, which in this example is "myHtmlProperty". Note the inclusion of a required XHTML Schema **xmlns** namespace declaration and **SchemaLocation** in the first ** ** element. They

[46] **Property** names, value enumerations and display behavior should be defined in use-case-specific profiles, FF software and style sheets. Proper implementation of **Property** content and display behavior should take place during the quality testing phase as part of the form modeling workflow. By design, the SDC Schema does not enforce **Property** element content beyond the basic **Property** structure defined by the SDC Schema. This allows **Property** customization for special use cases. Schematrons may be used to enforce XML patterns for custom **Property** types.

[47] Attribute text and XHTML strings subsumed by **Property** elements should preserve whitespace when rendered in a DEF, and special characters that disrupt XML and processing must be escaped out using standard XML escaping rules. These rules apply to any text in any SDC attribute.

[48] To ensure proper rendering in all FFs, formatted XHTML should generally always have equivalent non-formatted **title** text as well. XHTML text must use the XHTML namespace and Schema. The decision to display **Property/HTML** text depends on the rendering capabilities of the FF.

are both required for the use and validation of a **Property** with strongly-typed XHTML content.[49] As noted above, the **HTML** datatype element is used for this purpose, but the validation is accomplished with an XHTML Schema.

```
<Property propName="myHtmlProperty" val="This is plain property text">
    <TypedValue>
        <HTML>
            <div    xmlns="http://www.w3.org/1999/xhtml"
                    xsi:schemaLocation="http://www.w3.org/1999/xhtml xhtml.xsd">
            This is the <b>XHTML</b> version of the property text.
            </div>
        </HTML>
    </TypedValue>
</Property>
```

Example 13: **Property** *using* **TypedValue\HTML**

A strongly-typed **Property** supports all SDC datatypes. The next example shows a strongly-typed date **Property**, which enforces the proper date format. Note that **Property/@val** is missing in the **Property** element, an appears only in the date element. This is because **Property/@val** can only be validated against the *string* datatype, whereas **date/@val** validates using the W3C *date* datatype. Nevertheless, it is acceptable to repeat the same date in the **Property/@val** content. Modelers must ensure that the **Property/@val** content matches the strongly-typed **TypedValue/date/@val** content.

```
<Property propName="myDateProperty">
    <TypedValue>
        <date val="2019-01-01"/>
    </TypedValue>
</Property>
```

Example 14: **Property** *using* **TypedValue/date**

5.5.1.2 FormDesign/Property elements

As we saw earlier, The **Property** element may be used in conjunction with the **FormDesign** element:

```
<FormDesign xmlns="urn:ihe:qrph:sdc:2016" ID="Lung.Bmk.227_1.001.011.RC1_sdcFDF"
  baseURI="cap.org" fullURI="…" filename="Lung.Bmk.227_1.001.011.RC1_sdcFDF.xml"
  lineage="Lung.Bmk.227" formTitle="Lung Biomarker Reporting Template"
  version="1.001.011.RC1" xmlns="urn:ihe:qrph:sdc:2016"

  <Property propName="Copyright" val="(c) 2019 College of American Pathologists…"/>
  <Property propName="ApprovalStatus" val="RC1"/>

  <Body ID="Lung.Bmk.227_1.001.011.RC1_sdcFDF_Body"/>
</FormDesign>
```

Example 15: ***FormDesign/Property***

5.5.1.3 The reportText Property

The following example uses a **Question** XFC to introduce the **reportText Property**. We will cover the **Question** XFC in more detail later. **Question** text used for the DEF display is contained in the **Question/@title** attribute. In the default case, this **Question/@title** text should also appear in a report with no changes. In some cases, however, the report text should be different than the DEF text. In this latter case, the **Question/@title** is used for display to the DEF user, but the **Property/@val** content (**"Gross Appearance:")** of the **reportText Property** is used for the report.

[49] We could instead have declared the xhtml namespace at the top of the XML document (using `xmlns:h="http://www.w3.org/1999/xhtml"`) and prefixed the xhtml elements with `h:` (i.e., using the "h" namespace prefix, `<h:div/>`) .

```
<Question ID="Q6" title="Describe the Tumor's Gross Appearance">
    <Property propName="reportText" val="Gross Appearance:"/>
    <ResponseField>
        <Response>
            <string val = "The tumor was of ovoid shape, fully encapsulated..."/>
        </Response>
    </ResponseField>
</Question>
```

Example 16: The `reportText` `Property`

Occasionally, we wish to hide some `title` text entirely from the report. This is achieved by placing "`{no text}`" in the `reportText` `Property` value as follows:

```
<Question ID="Q6" title="Describe the Tumor's Gross Appearance">
    <Property propName="reportText" val="{no text}"/>
    <ResponseField>
        <Response>
            <string val = "The tumor was of ovoid shape, fully encapsulated..."/>
        </Response>
    </ResponseField>
</Question>
```

Example 17: `Property` using "`{no text}`"

The `reportText` `Property` is useful on most XFCs, but especially for `Question` and `ListItem` text.

5.5.1.4 Other Property Types

As described earlier, it is also possible to use a special `Property` (e.g., "`titleHTML`") to specify rich (`HTML`) text for the DEF and/or report:

```
<Question ID="Q6" title="Describe the Tumor's Gross Appearance">
    <Property propName="titleHTML">
        <TypedValue>
            <HTML>
                <div xmlns="http://www.w3.org/1999/xhtml"
                    xsi:schemaLocation="http://www.w3.org/1999/xhtml xhtml.xsd">
                    Describe the Tumor's <b>Gross Appearance</b> </div>
            </HTML>
        </TypedValue>
    </Property>

    <Property propName="reportText" val="Gross Appearance:"/>
    <ResponseField>
        <Response>
            <string val="The tumor was of ovoid shape, fully encapsulated..."/>
        </Response>
    </ResponseField>
</Question>
```

Example 18: Rich text using `Property`/`TypedValue`/`HTML`

The next example demonstrates how a **Property** can define text that appears in a report, but should not appear in a DEF. By default, **DisplayedItem/@title** content should not appear on a report, because it is usually designed to aid the DEF user in filling out the DEF. However, custom text is sometimes needed on a report, but is not needed (by default) on the DEF because it can clutter up the screen and distract the user. This pattern is sometimes called a "report note," which uses the reportText **Property**. Note that the **DisplayedItem/@title** attribute in Example 19 could have been omitted form the XML entirely, since it is an optional attribute that contains no content ("").

```
<DisplayedItem ID="DI2" title="">
    <Property propName="reportText" val="I am a Report Note: ..."/>
</DisplayedItem>
```

*Example 19: Using the **reportText** **Property** on a **DisplayedItem** for reporting notes*

In Figure 3: The Data-Entry Form (DEF), we introduced the concept of *untitled Questions*, which are **Question**s that have no **title** content, but the **Question**'s subject is easily inferred from the context. However, in some cases, a particular DEF implementation style may need to display alternative text for the question. For these situations, alternative **title** text is provided in the form of the **altText Property**, as shown below:

```
<Question ID="Q7" title="">
    <Property propName="altText" val="This is alternative Question text"/>
    <!-- other Question XML elements omitted -->
</Question>
```

*Example 20: The **altText** **Property***

Properties can be used for any similar custom purposes on any EBT element. The **Property** names (i.e., the **propName** content) and implementation details must be agreed upon by form designers and implementors so that implementation code will function as expected.

5.5.2 The EBT Comment Element

The EBT also includes the ability to add optional **Comment** elements. The **Comment** element is defined by the CommentType, which inherits from BaseType. **Comment**s may be provided by the FDF designer, the FDF implementation code, or by the DEF user. The type and styling of the **Comment** (if present) can be customized with the **type** and **styleClass** attributes. The comments are simple ASCII text and may not include rich text or substructures. The **Comment** content is placed in the **val** attribute of the **Comment** element.

Examples of comments created by an FDF designer are: (1) messages *from the modeler* to the implementor, (2) internal, non-reported notes and messages from the *user* and (3) to qualify a *user* response where the FDF does not allow free-form text data entry. Comments from the DEF *user* might concern their response to a QR or the selection of a **ListItem**. If DEF users are allowed to add **Comment** elements, then the FDF implementation (the DEF) must include visual clues (e.g., a comment icon) in the DEF so that the user can add comments at appropriate points (e.g., on each **Question** and **ListItem**).

5.5.3 The EBT Extension Element

The **Extension** element provides a place to insert XML that is not defined in the SDC Schema. The **Extension** element is defined by ExtensionType, which inherits from BaseType. The sub-elements of **Extension** must provide a namespace (and ideally an XML Schema) for any non-SDC elements and attributes that are introduced. The **Extension** element provides almost infinite expansion flexibility for SDC. However, it also requires that form designers and implementers agree on the supported extensions, agree on which elements to accept and the contexts in which they may be used, document this usage, and create an extension validation mechanism to enforce correct usage and detect incorrect usage.

5.6 IdentifiedExtensionType (abstract):

The IdentifiedExtensionType (**IET**) adds two attributes to its parent Type, EBT. The two attributes allow the unique identification of XFCs in an FDF, and also allow unique identification of the FDF. The attributes are:

- `ID`: The ubiquitous `ID` attribute is a unique URI identifier for XFC types in **`FormDesign`**, and for **`FormDesign`** itself. It is required, and its uniqueness is enforced by the SDC Schema. URI identifiers are very flexible since they may assume any legal XML URI format.
- `baseURI`: The `baseURI` is required only in the **`FormDesign`** element; it is optional on XFCs. It identifies the organization that is responsible for designing and maintaining the FDF or XFC. If an XFC does not derive from the same organization as the default `baseURI` (the FDF `baseURI` which is defined in the **`FormDesign`** element), then a new value for the `baseURI` is entered on the XFC element to override the default one, and the new XFC `baseURI` is then inherited by all descendant XFCs unless overridden by a descendant XFC. In most cases, FDFs will contain only a single default **`FormDesign`** `baseURI` on the FDF, and no `baseURI` content in the XFCs.
- The XFC `baseURI` + `ID` should combine to form a globally unique identifier (CGUI), that uniquely identifies an FDF or an XFC in a particular FDF `lineage`. The same XFC `baseURI` and `ID` may be reused in derived or versioned forms, as long as the context stays the same, and any affected data elements remain unchanged in context and semantics. See section 4.4.3: FDF and XFC Identifiers (IDs) for more information.

5.7 DisplayedType

DisplayedType has two functions: it defines the **`DisplayedItem`** XFC for representing visible areas on the screen, and also acts as a building block for the other visible XFCs (**`Section`**, **`Question`**, **`ListItem`**, **`ButtonAction`**). DisplayedType defines most of the essential functional underlying capabilities of XFC-based controls. These capabilities include support for **`Link`**, **`BlobContent`**, **`CodedValue`**, and several fundamental **Event** and **Guard** types. **`Link`** contain the address of internal (DEF-based) or external (network-based) resources to support display or functionality in a DEF. **`BlobContent`** contains inline base-64-encoded Binary Large Objects (Blobs) of virtually any type, but primarily those defined as standard Media (MIME) Types. **Events** are DEF user actions that are handled by the DEF to alter DEF behavior or functionality in some customized way. The standard events are **`Enter`**, **`Exit`**, **`OnEvent`**. **`OnEvent`** is a generic event that is defined by the form designer. An SDC **Guard** is a unit of coded functionality that is activated or "fired" by one or more events *somewhere else* in the DEF. An SDC guard may be considered as the target of one or more events. The two built-in guards are **`ActivateIf`** and **`DeActivateIf`**. These guards activate and deactivate visible XFC-derived DEF controls based on events or states somewhere else in the DEF. The guards can test for certain conditions before they activate or deactivate the guard target.

DisplayedType attributes are inherited by **`DisplayedItem`**, **`Section`**, **`Question`**, **`ListItem`**, and **`ButtonAction`**:

- `title`: [O] [def: ""] [dt: string] The primary text to show on the form. Also known as "prompt" or "label" or "visibleText" or "caption"
- `enabled`: [O] [def: True] [dt: boolean] Determines whether the user can interact with the displayed item when the form is first displayed. All *disabled* items (`enabled` = "`False`") are treated as read-only: they are visible but may not be edited; they cannot fire events, but they do respond to the **`ActivateIf`** guard. The `enabled` content value is transitive to descendants, so that all descendants of a *disabled* parent are also disabled, regardless of their own `enabled` content value; however, descendants of an *enabled* parent behave according to their own `enabled` content value.
- `visible`: [O] [def: True] [dt: boolean] Determines whether the item should be visible on a computer screen when the form is first displayed. The `visible` content value is transitive to descendants, so that all descendants of an *invisible* (`visible` = "`False`") parent XFC are also *invisible*, regardless of their own `visible` content value; however, descendants of a *visible* parent behave according to their own `visible` content value.
- `mustImplement`: [O] [def: True] [dt: boolean] If this attribute is set to "`True`" (the default), then the form implementation (DEF) must make this item available for use (i.e. display them when appropriate) on the form.

- Child XFCs of a parent with **mustImplement** = "True" behave according to their own **mustImplement** content.
- If a parent XFC has **mustImplement** = "False", then the parent and all descendant XFCs may be omitted from the DEF as a single block of XML.
- If a parent XFC with **mustImplement** = "False" *is* implemented in a DEF, then all its descendants behave according to their own **mustImplement** content – that is, the child XFCs with **mustImplement** = "True" must be implemented in the DEF.

- **showInReport**: [O] [def: True] [dt: boolean] This attribute is used to omit blocks of XFCs from the FDF report. To omit text from a single XFC, use the reportText **Property**. The default is "True".
  - If **showInReport** = "False" on a **Question**, then the **Question** and its **ListItem**s (if any) and all descendants should be omitted from the report derived from the FDF.
  - If **showInReport** = "False" on a selected **ListItem** in a QS, then the entire question and all **ListItem**s and all descendants should be omitted from the report.
  - If **showInReport** = "False" appears on a selected **ListItem** in a QM, then the selected **ListItem** and all descendants should be omitted from the report. If no **ListItem** from the QM will appear in the report, then the QM **Question** text (the **title** or the reportText **Property** value) should be omitted as well.
  - If **showInReport** = "False" on a **Section**, then the entire **Section** and all **Section** contents (descendants) should be omitted from the report.
  - If **showInReport** = "False" on a **DisplayedItem**, then the **DisplayedItem** should be omitted from the report.

5.8 RepeatingType

This type represents XFCs that may be repeated based upon on the user's interaction with the form objects. Items derived from this type include **Section**s and **Question**s. The handling of **Section** and **Question** repeats will be discussed in section 0.

RepeatingType attributes include:

- **minCard**: [O] [def: 1] [dt: unsignedShort] The minimum number of repetitions allowed for a section or question. The user *must answer* any question that has **minCard** > 0. If **minCard** = "0", then the item and all descendent questions are optional to answer. If this attribute appears on a **Section** or **Question**, it indicates the minimum number of **Section** or **Question** repeats required for a form to be considered valid. **Section** and **Question** must contain at least one user response to be considered repeated. The use of **minCard** is undefined on a **Section** that has no **Question** content, and may be ignored, unless a usage is defined for a special use case. Optional. The default value is 1. Datatype = unsignedShort.
- **maxCard**: [O] [def: 1] [dt: unsignedShort] The maximum number of repetitions allowed for a **Section** or **Question**. The default content value is 1, indicating that the **Section** or **Question** cannot be repeated on the data entry form. A content value of 0 indicates that the number of repeats is unlimited. If **maxCard** is not 0, then **maxCard** **must** be greater than or equal to **minCard**. The use of **maxCard** is undefined on a section that has no **Question** content, and may be ignored, unless a usage is defined for a special use case. Optional. The default value is 1. Datatype = unsignedShort.

- `repeat`: [O] [def: 0] [dt: nonNegativeInteger] Represents the repeat ordinality in an FDF-R, starting with `0`. `0` is always used for the original **`Question`** or **`Section`**, not a repeated one. Optional. Datatype = unsignedShort.

- `instanceGUID`: [O] [def: ""] [dt: string] A globally unique string assigned to a repeating **`Question`** or **`Section`**, **`InjectForm`**, or **`ListItem`**.[50] This attribute's value is assigned at the time that answers are entered into a form, to unambiguously globally identify a single instance among **`Section`** or **`Question`** elements, including those that are allowed to repeat and nest deeply. This provides a single globally-unique identifier for user-responses, and is a component of a child → parent linked list used for FDF data transmission.

- `parentGUID`: [O] [def: ""] [dt: string] A globally unique string, assigned on a **`Section`**, **`Question`**, **`ListItem`** or **`InjectForm`**, which contains the `instanceGUID` of its parent XFC node (**`Section`**, **`Question`**, **`ListItem`** or **`InjectForm`** only[51]). This attribute's value is assigned at the time that answers are entered into a form. Assignment of `parentGUID` results in the creation of a child-parent linked list GUID tree among XFCs that may repeat and nest deeply. This helps to preserve the exact context of the FDF-R responses when the data are persisted in a database or other data store.

[50] Since **`Section`** and **`InjectForm`** usually contain child **`Question`** elements and thus may contain user-entered instance data, assignment of GUIDs to them is helpful. **`ButtonAction`** and **`DisplayedItem`** are not central to the captured data content of a DEF, and they do not have the ability to subsume **`Question`**s. Therefore, assigning GUIDs to repeating instances of BA and DI will not help in determining data context in an FDF-R.

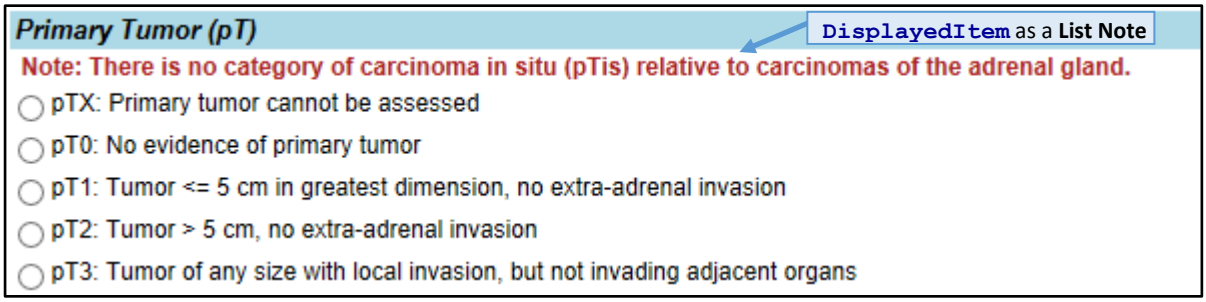
[51] **`ButtonAction`** and **`DisplayedItem`** cannot have child XFCs, so these will never appear as a parent to another XFC.

6 The XFCs and their DEF Representations

6.1 The `DisplayedItem` XFC

As described above, DisplayedType, which provides the **`DisplayedItem`** (DI) definition, is the parent Schema Type of the XFC Types for S, Q, LI, and BA. Thus, S, Q, LI, and BA inherit all elements and attributes of DisplayedType. **`InjectForm`** does *not* inherit from DisplayedType, as it relies on the display properties of the subsumed injected parts, which *do* inherit from DisplayedType.

The primary function of DI is to provide text in the DEF. Often this takes the form of "notes," which are packets of DEF text that may accompany QAS controls to provide explanatory information with background documentation. Although DEF notes usually derive from the DI XFC, they can also derive from other SDC objects such as **`Section`** titles and **`Property`** values. The DI (as well as all DisplayedType descendants) can additionally be used to display Links (e.g., to Internet resources), **`BlobContent`** (Binary Large Objects) of any Media (MIME) type, **`Contact`** information and **`CodedValue`**s.




```
<Question ID="2137.100004300" title="Primary Tumor (pT)">
  <ListField>
    <List>
      <DisplayedItem ID="20894.100004300" title="Note: There is no category of carcinoma in situ (pTis) relative to
      carcinomas of the adrenal gland.">
        <Property type="reportText" val="{no text}" />
      </DisplayedItem>
      <ListItem ID="2142.100004300" title="pTX: Primary tumor cannot be assessed">
        <Property name="pTX_346" type="reportText" val="pTX" />
      </ListItem>
```

`DisplayedItem`: functions as a note inside a **`List`** of **`ListItem`**s

A **`DisplayedItem`** has a unique **`ID`** and a **`title`**, but unlike **`Questions`** and **`Sections`**, cannot have a **`ChildItems`** element or descendant XFC form components. **`DisplayedItem`**s that must have XFC descendants should use the **`Section`** element instead.

6.1.1 `DisplayedItem` Substructure

6.1.1.1 `BlobContent`

Blobs can handle any type of base-64-encoded binary media, or a link to binary media. Binary media may include images, audio, video, etc.

The generic XPath for **`BlobContent`** is **`DisplayedItem\BlobContent`**.

`BlobContent` child elements are:

- **`Description`**: Description of the media (optional)
- **`Hash`**: Binary hash of the blob data (optional)
- **`BlobURI`**: For links to a binary resource (one of either **`BlobURI`** or **`BinaryMediaBase64`** is required)

- **BinaryMediaBase64**: For inclusion of base-64-encoded binary data inside the FDF

BlobContent also introduces two new attributes:

- **mediaType** is required and should be derived from a standard nomenclature such as a Media (MIME) type).[52]
- **fileExtension** is optional and indicates the filename extension if the (base-64-encoded) blob were to be decoded to its native binary format and saved as a disk file. For example, it could be "jpg" or "png" for image formats.

6.1.1.2 Link

An SDC **Link** has the same function as a web browser link. In can display linked content in several ways, including inline or pop-up displays, new window/tab, etc. Link has two child elements: **LinkText** (optional) and **LinkURI** (required) with functions identical to the HTML counterparts. The **LinkURI** may contain **name** content derived from BlobContent within the same FDF.

6.1.1.3 Contact

Some use cases may benefit from the inclusion of person or organization metadata attached to FDF content. **Contact**, which is derived from ContactType, has two optional child elements: **Person** and **Organization**. Each of these elements has a deep substructure. The SDC Schema should be consulted for more detailed information.

6.1.1.4 CodedValue

The **CodedValue** element is mostly useful to DisplayedType descendants (**Section**, **Question**, **ListItem**, **ButtonAction**) rather than **DisplayedItem** itself. It provides hidden metadata to include a code, terminology, classification, keyword, or local value that may be necessary in certain use cases. SDC prefers the use of externally maintained code maps to codes or similar references inside the FDF. This is because codes generally need more maintenance than the rest of the FDF, resulting in the need for excessive versioning of the FDF due to coding updates. **CodedValue** may also be used to insert values that are useful for computer processing inside the DEF. (Also see **associatedValue** in the section on ListItem Attributes.)

CodedValue derives from EBT, and has no custom attributes. It has the following child elements:

- **Code**: A standard code, or a local value from a custom coding system, that can be used to consistently identify, or provide a standard value for, the coded item.
- **TypedValue**: Data type enumeration derived from W3C XML Schema. If the code is derived from a local value system (e.g., numbered answer choices such as clock positions, tumor grades, or clinical scoring systems), then the data type of the local value may be specified here. This may be important if the code value will need to be manipulated mathematically.
- **CodeText**: The human readable text that accompanies the assigned code and represents the code's precise meaning (semantics) or usage. For example, this element could contain the SNOMED CT Fully-Specified Name (FSN).
- **CodeMatch**: Degree of match between the mapped item (e.g. the XFC **title** content shown in the DEF) and the assigned code. Its **codeMatchEnum** attribute holds an entry from an enumerated list of match types. Note that some of the list entries are only appropriate for terminology codes, while others are only appropriate for DEs:
    - Exact Code Match
    - Close Code Match
    - Code Broader Than Item

[52] See https://en.wikipedia.org/wiki/Media_type

- Code Narrower Than Item
- Item Implements Data Element Exactly
- Item Derived from Data Element
- Item Related to Data Element

- **CodeMatchComment:** Comment about the degree of match between the mapped item and the assigned code.

- **CodeSystem**: An EBT-derived element structure that defines the system that creates and maintains the standards for the code map. Its child elements are:
  - **CodeSystemName**: The name of the coding system, as recommended by the coding system curators, or as recommended by the agency that creates standards for the code map in use.
  - **ReleaseDate**: The day that the selected version of the coding system was released for general use by the coding system curators.
  - **Version**: Version of the coding system; uses the version format defined by the coding system.
  - **OID**: The ISO object identifier (OID) for the coding system, as found at the HL7 OID Registry: https://www.hl7.org/oid/index.cfm
  - **CodeSystemURI**: Web resource that uniquely identifies the coding system.

DefaultCodeSystem on **ListField** may be used with code systems to assign a default code and code type if all **ListItems**s in a **ListField** are associated with coded values from a single coding system. **DefaultCodeSystem** and **CodeSystem** are both defined by **CodeSystemType**, and thus have identical structure.

```
<Question title="Procedure protocol:" ID="000021">
    <CodedValue>
        <Code val="RID38760" />
        <CodeText val="imaging protocol" />
        <CodeSystem>
            <CodeSystemName val="RADLEX" />
        </CodeSystem>
    </CodedValue>
    <ListField>
        <List>
            <ListItem title="LDCT Study Protocol" ID="000022">
                <CodedValue>
                    <Code val="RPID1377" />
                    <CodeText val="CT CHEST LOW DOSE SCREENING" />
                    <CodeSystem>
                        <CodeSystemName val="RADLEX" />
                    </CodeSystem>
                </CodedValue>
            </ListItem>
            ... (closing tags)
```

CodedValue

CodedValue

*Example 21: **CodedValue***

6.1.2 **DisplayedItem** Events and Guards (TBD)

- **OnEnter**: Fires when user clicks, moves or tabs into the DEF screen area of an XFC
- **OnExit**: Fires when user clicks, moves or tabs out of the DEF screen area of an XFC
- **OnEvent**: Generic event handler – the **eventName** must be specified and documented based on the use case
- **ActivateIf**: Activate the DisplayedType item if the guard conditions are met
- **DeactivateIf**: DeActivate the DisplayedType item if the guard conditions are met

6.2 The Section XFC

The **Section** element derives from RepeatingType (section 5.8). It introduces one new element, **ChildItems**, and no new attributes. A Section contains a required **ID** and an optional **title**. It subsumes a group of topically-related XFCs, which translate to DEF controls. A **Section** can contain any number of sub-**Section**s and other XFCs under its optional **ChildItems** element, except for **ListItem**s. (**ListItem**s may appear in a **Section**, but must be within a **Question**/**ListField**/**List**/**ListItem** structure.)

Attributes:

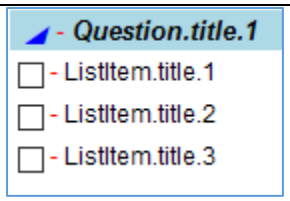
- **ordered**: If false, then the form implementation may change the order of items in the section.

Response Attributes: **newData**, **changedData**, **approvalStatus** and **completionStatus** are documented under **FormDesign**.

6.3 The Question XFC

The following figure depicts a simple multi-select **Question** (QM)

```
<Question ID="QM1" title="Question.title.1" >
    <ListField maxSelections="0">
        <List>
            <ListItem ID="LI1" title="ListItem.title.1" />
            <ListItem ID="LI2" title="ListItem.title.2" />
            <ListItem ID="LI3" title="ListItem.title.3" />
        </List>
    </ListField>
</Question>
```



Example 22: Question XFC XML

There are three basic QAS types: **Question-Response** (QR), Single-Select **Question** (QS), and Multi-Select **Question** (QM). These are covered next.

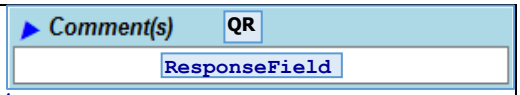
6.3.1 Question-Response (QR)

The **QR** XFC does not have an answer list (a **List** of **ListItems**).[53] The QR control represents a **Question** control with an adjacent area for capturing a response from the user. The response area is defined by a **ResponseField** element structure.

A QR response may be validated by the substructure of the **Response** element, which determine if the inputs are in the proper format (e.g., as a text string or decimal, or rarely, following a pattern mask or a binary type).

The following FDF XML sample contains the basic parts of a QR XFC:

```
<Question ID="40273.100004300" title="Comment(s)">
    <ResponseField>
        <Response>
            <string maxLength="4000" minLength="0"/>
        </Response>
    </ResponseField>
</Question>
```



Example 23: QR XFC XML

Observe the following points about Example 23:

- **Question** is the only XFC that appears in this XML example. The other nested elements support the **Question** XFC with additional metadata.
- All XFC elements (**Question** in this case) must have an **ID** that is unique within the FDF which enforced by the XML Schema. Uniqueness of IDs must be confirmed by SDC-compliant software. Since XFCs can be

[53] The QR is sometimes called a "Question-Fill-in" (QF).

repeated in DEF implementation, uniqueness-enforcement is an important task in SDC software implementations.

- The text to display in the DEF is in the `title` content.
- Responses are captured in a **`ResponseField/Response`** / *(datatype)* structure, where *(datatype)* is an element corresponding to one of the SDC DEtype datatypes.[54]
    - **`Response`**s must contain an SDC datatype element (**`string`** in this case).
    - User-entered responses are captured in the `val` attribute of the *(datatype)* element, like this:

      ```
      <string maxLength="4000" minLength="0" val="This is my comment" />
      ```

The **`Response`** field can also handle rich text from user input. In the next example, the **`Response/string`** datatype is replaced by the **`HTML`** SDC datatype, allowing the DEF user to enter HTML text into the DEF instead of unformatted text:

```
<Response>
    <HTML>
        <div xmlns="http://www.w3.org/1999/xhtml"
             xsi:schemaLocation="http://www.w3.org/1999/xhtml xhtml.xsd">
             <b>The tumor was of ovoid shape, fully encapsulated</b>
        </div>
    </HTML>
</Response>
```

Example 24: Using `Response`/`HTML` text

Note the use of the xhtml namespace and the `xsi:schemaLocation` that is used for validation of the XHTML content.

Here is another example that shows how numeric QR structures may appear:

```
<Question ID="QR1" title="Percentage of Necrosis">
    <ResponseField>
        <Response>
            <integer maxInclusive="100" minInclusive="0"/>
        </Response>
        <TextAfterResponse val="%"/>
        <ResponseUnits val="%" unitSystem="UCUM"/>
    </ResponseField>
</Question>
```

Example 25: QR with Numeric ResponseField

Observe the following points:

- This example uses an **`integer`** SDC datatype to specify and record the user-entered value.
- When the user enters a response into the DEF, the **`integer`** element's `val` attribute will be populated. The `val` attribute is strongly-typed in the SDC Schema according to its parent element's type (**`integer`**), and can thus be used to partially validate the `val` content:

  ```
  <integer maxInclusive="100" minInclusive="0" val="25"/>
  ```

- The displayed datatype validation metadata are specific for the **`integer`** datatype (`maxInclusive`, `minInclusive`), and are also strongly-typed to match the parent element's datatype (**`integer`**).
- Other attributes are available for numeric response datatypes, including: `quantEnum`, `maxExclusive`, `minExclusive`, `totalDigits`, `mask`, `allowGT`, `allowGTE`, `allowLT`, `allowLTE`, and `allowAPPROX`.

[54] SDC supports W3C datatypes and also supports XML and HTML datatypes for user responses.

For decimals and float, `fractionDigits` is also available. These additional validation attributes are strongly-typed according to the parent element's datatype, and can be used to validate user responses in the strongly-typed `val` attribute. Validation of `val` content may be achieved by using a Schematron or other programming technique.

- The remaining **`ResponseField`** elements and attributes are described below in section 6.3.1.1.1.

6.3.1.1 `ResponseField` (RF) and `ListItemResponseField` (LIRF) Elements

The **`ResponseField`** element is found only on a QR **`Question`**. It is defined by ResponseFieldType, which derives from EBT.

The **`ListItemResponseField`** is an element under **`ListItem`**. It has a structure that is identical to **`ResponseField`**, except for the addition of a single attribute:

- `responseRequired`: If `responseRequired` is set to true, then the appropriate data must be entered in the data-entry field associated with the **`ListItem`**. Datatype is boolean. Default is "`False`".

The **`ResponseField`** and **`ListItemResponseField`** elements are where user input is stored in an FDF-R. Their substructure and metadata determine the rules for allowed data entries.

6.3.1.1.1 `ResponseField` Sub-Elements

- **`Response`**: This element holds several sub-elements for specifying the display of captured answers as well as for constraining data entry by a user. The user's data is entered into a strongly-typed sub-element that is typed according to any SDC datatype. The available datatypes are defined by DataTypes_DEType.
- **`TextAfterResponse`**: This element provides a place to specify static text that should appear *after* the user's response in the DEF and the report. The `val` content holds the text for display in the DEF and report. The `val` content appears in the DEF after (to the right of) the user's response on the data entry form. This may be text for units such as "mm", "cm", etc.
    - **`RichText`**: Representation of plain text user content (`val`) with an option for HTML-formatting. Contains optional boilerplate metadata to aid programmatic manipulation.
- The **`ResponseUnits`** element provides a place to record the units relevant to the response. The `val` attribute holds the content ("`%`") of the units for the user's response. This is provided by the FDF designer, not the DEF-user. The **`ResponseUnits`** content is "hard-coded" into the FDF by the FDF modeler, and is not changeable via the DEF. The `unitSystem` attribute defaults[55] to "`UCUM`" (Unified Code for Units of Measure) and is not usually displayed in the DEF or report.

6.3.1.1.2 `ResponseField` Events

- **`AfterChange`** event: event that occurs after the **`Response`** value is changed, usually fired after a user leaves the Response field.
- **`OnEvent`** event: Adds a custom event handler to a form item such as a question, section or list item. `eventName` is required.

6.3.1.1.3 The `Response` Element and its Datatypes

- The Response Element subsumes any one of the SDC datatype elements as defined in DataTypes_DEType,

55 All default XML attribute values may be omitted from the XML, but should be treated as if they were present, holding the default value.

e.g., **string**, **integer**, **decimal**, **dateTime**, **duration**, **HTML** or **XML**.

- Almost all SDC datatype element names use camelCase.[56] This camelCase element format is unlike all other elements in the SDC Schema, which use PascalCase. The camelCase format is used to conform with the W3C representation of the datatypes in XML Schema documents. The only exceptions to the camelCase format are **XML** and **HTML**.
- All datatypes are defined and modeled after the W3C datatypes and include all of the W3C datatype metadata attributes.
- All metadata attributes in an SDC datatype element have a type appropriate for the element and the metadata. For example, **maxLength** has an XML Schema datatype of xs:long.[57]
- The **val** attribute is strongly-typed and thus can be used to validate the datatype of its element tag. Entering an invalid datatype in this attribute will generate a validation error when validated with the SDCFormDesign Schema.
- **val** cannot be Schema-validated against the element's other datatype metadata (e.g., for **string**, these would include: **maxLength**, **minLength**, **mask**, and **pattern)**, so that further validation of **val** content must use additional methods such as Schematron.

[56] i.e., they begin with a lower-case letter, but each new word part is capitalized and appended to the first word part, e.g., "**nonNegativeInteger**", "**integer**", "**gYear**", etc. (Almost all SDC *attributes* use this format. The one notable exception, because of its importance, is **ID**.)

[57] "xs" is a common namespace prefix for XML Schema, and refers to the fact that the datatypes are defined in the XML Schema documentation.

6.3.2 The Single Select `Question` (QS)

A **QS** control displays a `Question` that subsumes a group of `ListItem` controls in a `ListField/List/ListItem` structure. The `List` is set to single-select in the `ListField` metadata and in the DEF control. The QS control is often rendered as a `Question` title area along with option/radio buttons for the LIs, or as a combo-box with dropdown answer choices, from which only one answer may be selected. The QS `ListField` element has `maxSelections="1"` which is the default value for this attribute; if `maxSelections` does not appear in the `ListField` (as in the Example 26 below), then the `Question` is treated as a QS. In the example, note that the `ListItem` with `title = "Blue"` has been selected (`selected = "true"`). Also note the presence of a `reportText` `Property` and a `ListItemResponseField` field under the last `ListItem`.

```
<Question ID="QS1" title="Select your favorite color ">
    <ListField>
        <List>
            <ListItem ID="LI1" title="Red"/>
            <ListItem ID="LI2" title="Green"/>
            <ListItem ID="LI3" title="Blue" selected="true"/>
            <ListItem ID="LI4" title="Black"/>
            <ListItem ID="LI5" title="Enter another color" >
                <Property propName="reportText" val="{no text}"/>
                <ListItemResponseField responseRequired="true">
                    <Response>
                        <string maxLength="30" />
                    </Response>
                </ListItemResponseField>
            </ListItem>
        </List>
    </ListField>
</Question>
```

ListField: maxSelections="1" is not shown, since it is the default value

Example 26: The Single Select `Question` (QS)

6.3.3 The Multi-Select `Question` (QM)

A **QM** control also displays a `Question` that subsumes a `List` of `ListItems`. The `ListField` element that subsumes the `List` element is set to multi-select by setting `maxSelections` to `0` (no limit on number of selections), or to a value greater than 1 (if there is a limit to the number of selections allowed). The control is often rendered with *checkboxes* for LIs, from which more than one `ListItem` may be selected. These checkboxes may be part of a combo-box/dropdown control as well.

```
<Question ID="20862.100004300" title="Tumor Description" mustImplement="false" minCard="0">
  <ListField maxSelections="0">
    <List>
      <ListItem ID="20863.100004300" title="Hemorrhagic" mustImplement="false" />
      <ListItem ID="20864.100004300" title="Necrotic" mustImplement="false" />
      <ListItem ID="17883.100004300" title="Invasion" mustImplement="false">
      ... (closing tags)
```

ListField: `maxSelections` = "0" means this is a QM, but QM also could have any number greater than 1

Example 27: The Multi-Select `Question` (QM)

As described earlier, there are two `ListItem` types:

- An **LI** control that displays a simple `ListItem`, as described above

- An **LIR** control displays a **`ListItem`** with an associated visible control for the **`ListItemResponseField`** to capture the user's fill-in response, similar to a QR.

```
<Question ID="20869.100004300" mustImplement="false" minCard="0">
  <ListField maxSelections="0">
    <List>
      <ListItem ID="20870.100004300" title="Capsule" mustImplement="false" />
      <ListItem ID="20871.100004300" title="Vessels" mustImplement="false" />
      <ListItem ID="20872.100004300" title="Extra-adrenal (specify)" mustImplement="false">
        <ListItemResponseField name="Extra_Adrenal_192" responseRequired="true">
          <Response>
            <string name="Extra_Adrenal_194" maxLength="4000" />
            ... (closing tags)
```

LIR

Example 28: A `ListItem` with a `ListItemResponseField` (LIR)

6.3.4 Capturing User Responses in `Question`s

Capturing data in a **`Question`** can occur in two ways:

- **Response**: A response (answer) to a QR or LIR is captured directly into the FDF **`Response`** element in the strongly-typed `val` attribute. Most of the other available attributes in the SDC datatype element (**`integer`** in this case) are also strongly-typed, and can be used to help validate the user response in `val`. In this example, `val` is typed as integer, and will generate an SDC Schema validation error if non-integer content is entered.

QR

```
<Question ID="q2" title="My Question">
  <ResponseField>
    <Response>
      <integer val="123"/>
    </Response>
  </ResponseField>
</Question>
```

Example 29: Capturing User Responses in a QR `Question`

LIR

```
<Question ID="q1" title="Question-1">
  <ListField>
    <List>
      <ListItem ID="341" title='ListItem-1' selected="true">
        <ListItemResponseField responseRequired="true">
          <Response>
            <string val="My string response" />
          </Response>
        </ListItemResponseField>
      </ListItem>
      ...
    </List>
  </ListField>
</Question>
```

Example 30: Capturing User Responses in a QS `Question`

- **Selection:** Each selected **`ListItem`** on a QS or QM is captured by setting the `selected` attribute on the **`ListItem`** element to **`"True"`**. If the `selected` attribute does not appear, it takes its default value, which is `'False'`.

```
<Question ID="q1" title="Question-1">
  <ListField>
    <List>
      <ListItem ID="341" title='ListItem-1' selected="True"/>
    </List>
  </ListField>
</Question>
```

Example 31: Selecting a `ListItem` in a QS `Question`

The FDF may contain **`Question`** XFCs that are pre-configured to contain *default responses* by setting **`selected`** to "`True`" or providing **`Response`** data in the **`val`** attribute of a **`Response/{SDC datatype}`** element. Default responses are set in the FDF XML as illustrated in the prior examples and are displayed in the DEF upon loading the form.

6.4 The `ListField` Element

6.4.1 `ListField` Attributes:

- **`colTextDelimiter`**: [O] [def: "|"] [dt: string] Character in the title that separates the columns and rows in a single or multi-column list.
- **`numCols`**: [O] [def: 1] [dt: unsignedByte] Number of columns in the list.
- **`storedCol`**: [O] [def: 1] [dt: unsignedByte] Determines which column of the list is stored in a database. This list is one-based.
- **`minSelections`**: [O] [def: 1] [dt: unsignedShort] Minimum number of answer choices (**`ListItem`**s) that must be selected by the user. Minimum value is 1.
- **`maxSelections`**: [O] [def: 1] [dt: unsignedShort] Maximum number of answer choices (**`ListItems`**) that can be selected by the user. Must be greater than or equal to **`minSelections`**, and no larger than the total number of list items.
  A value of 0 indicates no limit to the number of selected **`ListItem`**s. This effectively means that the **`Question`** is multi-select. (Abbreviated as QM). A value of 1 (the default) indicates that the **`Question`** is single-select. (Abbreviated as QS)
- **`ordered`**: [O] [def: True] [dt: boolean] If false, then the form implementation may change the order of items in the list.
- **`defaultListItemDataType`**: [O] [dt: DataTypeAll_StypeEnum] This attribute contains an SDC datatype enumeration. The selected value is the datatype of the content for all **`ListItem`**/**`associatedValue`** content in the current **`List`**. It is used instead of **`associatedValueType`**. This element is used only if the **`ListItem`**s are all associated with coded values from a single coding system. If **`associatedValueType`** on a ListItem has a datatype assigned, then the latter datatype overrides the content in **`defaultListItemDataType`**.

6.4.2 `ListField` Sub-Elements

- **`ListHeaderText`**: The header row for a set of list items. If the list has more than one column, the column text is separated by the colTextDelimiter.
- **`DefaultCodeSystem`**: If coded values are used for items in a **`List`** (including **`ListItem`** and **`LookupEndPoint`** lists), then the default coding system should be specified here. For **`ListItem`** nodes, any exceptions to the coding system may be specified in **`ListItem/CodedValue/CodeSystem`** on the individual **`ListItem`** nodes that contain exceptions. For **`LookupEndPoint`**s, the endpoint data can optionally specify any exceptions in a dedicated **`CodeSystem`** column in the returned list data. See CodeSystem for details.

- **IllegalListItemPairings**: This predicate rule specifies a set of **ListItem**s (**listItemNames**) that cannot be selected when a test **ListItem** (named in **testItemName**) is also selected. If any selection occurs in **ListItem**s listed in **listItemNames** when **testItemName** is selected, the rule evaluates to True. In all other cases, the rule evaluates to false. Multiple selections in **listItemNames** are acceptable as long as **testItemName** is unselected, and in this case, the rule evaluates to False. "Legal" (allowed) selections evaluate to False. "Illegal" selections evaluate to true. The predicate's Boolean value can be reversed if **not** is set to `True`. Also provides a **maxSelections** attribute: The maximum number of **ListItem**s in **listItemNames** that may be selected at one time (default = 1).
- **IllegalCoSelectedListItems**: This predicate rule tests combinations of co-selected **ListItem**s. The maximum number of allowed co-selected items, X, is specified in **maxSelections**, and the default is 1. If more than X items in the **listItemNames** list are selected, then the result returns the value of True. Otherwise it is False. The most common use is to detect **ListItem** combinations that may not be selected together. In most cases, all **ListItem**s should be children of one multi-select **Question**.

6.4.3 **ListField** Events

- **AfterChange** event: Event that occurs after **ListField** selections are changed.
- **OnEvent** event: Adds a custom event handler to a form item such as a **Question**, **Section** or **ListItem**. **eventName** content is required.

6.4.4 The **List** Element

6.4.4.1 **List** Sub-Elements

6.4.4.1.1 The **ListItem** XFC

6.4.4.1.1.1 **ListItem** Attributes

- **selected**: [O] [def: False] [dt: boolean] Selected is `True` if the **ListItem** is selected by the user, or if the default value of the **ListItem** is `True`. Otherwise it is `False` (the default).
- **selectionDisablesChildren**: [O] [def: False] [dt: boolean] If set to `True`, then selecting this ListItem must deactivate all descendant parts of the form, and ignore any user-entered values in the deactivated part. Deselecting the **ListItem** should reactivate the descendant items in their state at the time the items were deactivated.
  If items are disabled, then any data stored in the disabled questions should be removed. Default is `False`.
- **selectionActivatesItems**: [O] [def: False] [dt: boolean] Selecting the current **ListItem** will enable the named items in this attribute's content. Prefixing any named with a hyphen (-) will reverse the above behavior (i.e., the named items will be disabled). Unselecting the **ListItem** will reverse this behavior. Prefixing the name with a tilde (~) will suppress this reversal behavior.
- **selectionSelectsListItems**: [O] [def: ""] [dt: NMTOKENS] Selecting the current **ListItem** will select the named **ListItem**s in this attribute's content. Prefixing any named with a hyphen (`-`) will reverse the above behavior. Unselecting the **ListItem** will reverse this behavior. Prefixing the name with a tilde (`~`) will suppress this reversal behavior.
- **selectionDeselectsSiblings**: [O] [def: False] [dt: boolean] If the ancestor **ListField** has **multiselect=**"`True`", then selecting this **ListItem** should de-select all other **ListItem** (sibling) nodes except the current one.
- **omitWhenSelected**: [O] [def: False] [dt: boolean] If **omitWhenSelected** is set to `True`, then the question and its response(s) should not be present in a typical report derived from this template. This attribute is usually set to true when the answer choice is used to control form behavior (e.g., skip logic), or

when the question provides unhelpful "negative" information about actions that did not occur or were not performed, or things that were not observed or could not be assessed. If `omitWhenSelected` is set to false (default) then the question and its response(s) should appear in the report.

- `associatedValue`: [O] [def: ""] [dt: string] A value (e.g., an integer) that is uniquely associated with a `ListItem`. An example is the integer 10 for a `ListItem` with a `title` that reads "10 o'clock". Typically, these values are set to be used in calculations or other algorithms. In general, they can be treated something like a user-entered response on a the `ListItemResponseField` of a selected `ListItem`. This field should not be used for terminologies or local codes. The `CodedValue` type should be used for these kinds of metadata. Also, this field should not be used for other metadata such as translations or usage. The datatype should be specified in `associatedValueType` or `ListField/@defaultListItemDataType`.
- `associatedValueType`: [O] [def: "|"] [dt: DataTypeAll_StypeEnum] The datatype of `associatedValue`, chosen from the DataTypeAll_StypeEnum enumeration in the SDC Schema. The content of this attribute overrides `ListField/@defaultListItemDataType`.

6.4.4.1.1.2 `ListItem` Sub-Elements

6.4.4.1.1.2.1 `ListItemResponseField` (LIRF) Element

LIRF uses the same elements as `ResponseField` ; See here for details. LIRF adds one new attribute:

- `responseRequired`: [O] [def: False] [dt: boolean] If `responseRequired` is set to true on a selected `ListItem`, then the appropriate content (e.g. text, numeric, or media type) must be entered in the data-entry field associated with this list item.

6.4.4.1.1.3 `ListItem` Events and Guards (draft)

- `OnSelect` event: Fired when the `ListItem` is selected.
- `OnDeselect` event: Fired when the `ListItem` is deselected.
- `SelectIf` guard: Selects the `ListItem` when the guard evaluates to True.
- `DeselectIf` guard: Deselects the `ListItem` when the guard evaluates to True.

6.4.4.1.2 The `DisplayedItem` inside of a `List` Element (List Notes) (TBD)

6.4.5 The `LookupEndpoint` Element (draft)

`LookupEndPoint` is used when the list items are derived from a web service call of some type, instead of an explicit set of `ListItem` nodes specified in the `FormDesign` XML. The endpoint function must return a list separated into individual list items by the `colTextDelimiter` value specified in the parent `ListField`. `includeHeaderRow` is a Boolean flag that determines how to use the first returned row, i.e., as a `ListItem` surrogate or as a header row. The `ResponseValue` element stores the user's response to the lookup list. The response is recorded as a local value (with a specified datatype) or as a coding, terminology, classification, or keyword. Multiple selections from the lookup list may be allowed.

6.5 The `ButtonAction` XFC (draft)

A Button is a visible area that can be selected ("clicked") to trigger event code to perform an action. It is ordinarily implemented with an image that looks like a rectangular button object, but a button can take on the appearance of any image or screen area.

The ButtonAction type represents a visual area for a user to click, and the click triggers a predicate expression to run "Action" code that runs inside the form. The ButtonAction object may be represented with a visible button object, or some other type of visual paradigm. (Other types of actions, e.g. key presses in a text field, may be handled with the form framework's event model.) It includes a single new element, the `OnClick` Event, which is an unspecialized EventType structure.

6.6 The `InjectForm` XFC (draft)

Elements: `InjectForm` is a placeholder indicating the location where parts of the current or a different FDF are to be "injected" DEF. `InjectForm` may subsume only one injected element – either `Section`, `Question` or `FormDesign`. These three top-level elements may subsume any content allowed by the SDCFormDesign Schema.

`InjectForm` introduces the following attributes to determine the injected element at runtime:

- `pkgFullURI`: [O] [def: ""] [dt: anyURI] The injected package is retrieved form `pkgManagerURI` + "/" + `pkgFullURI`. If pkgFullURI is null, then then current form is used injection.
- `PkgManagerURI`: [O] [def: ""] [dt: anyURI] The server from which the injected package will be retrieved.
- `rootItemID`: [def: ""] [dt: anyURI] The `ID` of the form or form part that will be injected. It must point to a valid FormDesign, Section or Question element.

7 DEF Functional Considerations

7.1 The DEF Maintains and Manipulates the FDF

The DEF, running inside a FF application, should maintain a copy of the FDF and manipulate its XML to add the user's responses, captured by the DEF, into the XML. This process involves inserting the captured responses into specific XFCs in the FDF. FDF XML manipulation can occur with each user interaction and/or at the time of DEF data submission to the Form Receiver.

7.2 Implied Activation (IA)

The **visible** and **enabled** attributes on XFCs may be used to explicitly set these XFC appearance and behavior features when an FDF is loaded and/or manipulated by the user. An XFC with **visible** = "False" cannot be made visible without an explicit SDC rule or programming code that resets **visible** to "True". An XFC with **enabled** = "False" cannot be enabled without an explicit SDC rule or programming code that resets **enabled** to "True". By default, both of these attributes are set to "True" on every XFC, and therefore these attributes can be ignored in most cases.

However, the *implied activation* of an XFC (that has **visible** and **enabled** set to "True" by default) is controlled by the user's entering of data into the DEF.

An "activated" DEF item means that a user can interact with that item. If the active item is a **Question**, then the user can answer it. Thus, an active item is both *visible* and *enabled* in the DEF.

Conversely, a *de*activated item is disabled and/or invisible (regardless of the **visible** and **enabled** attribute content), so that a user cannot interact with it in the DEF. If an item is deactivated, then all its descendants are also deactivated. (The only exception is provided with **selectionDisablesChildren**, described later.)

A deactivated item must not be subject to a response validation procedure, should not appear on a report, and should not be saved in a database, even if it contains a deactivated response from earlier user activity in the form. Such earlier activity in a previously-activated item is now overridden by the deactivated status of the same item.

The hierarchical arrangement of items in the FDF file, as reflected in the DEF, allows us to define a drill-down item activation pattern as the user interacts with DEF items. This drill-down behavior can affect the layout, workflow, validation, reporting and data storage properties of DEFs.

In Example 32A, **ListItem** LI.1b is the parent of **QS2** in the XML and in the DEF. **QR3** is the parent of **QS4.**

The DEF will open with **QS1** and **QR3** activated, but **QS2** and **QS4** will appear inactive (disabled).

Implied Activation (A)	Implied Activation (B)	
QS1: ○ LI.1a ○ LI.1b **QS2** ○ LI.2a ○ LI.2b **QR3: _____** **QS4** ○ LI.4a ○ LI.4b	**QS1** ○ LI.1a ○ LI.1b **QS2** ○ LI.2a ○ LI.2b ○ LI.2c	**Key for Examples:** Green = Parent Question Blue = ☐ ListItem (multi-select) ○ ListItem (single-select) Red = Child (dependent) Question

Example 32: Implied Activation

In general, when opening an eCC DEF for the first time, only the outermost level of the DEF item tree should be activated.[58] As the user answers **Question**s, new parts of the item tree become activated. This "Implicit Item Activation" occurs when a selected **ListItem** or an answered QR activate successive layers of dependent (child) items.

If LI.1b is selected, then **QS2** will become activated. When **QR3** receives a fill-in response, then **QS4** will be activated.

If the parent **QS1** and **QR3** responses are removed, then the activation of the child items is reversed (they become deactivated again). If the parent **QS1** and **QR3** items become deactivated by *their* parent items, then all their descendants (including **QS2** and **QS4**) also are deactivated.

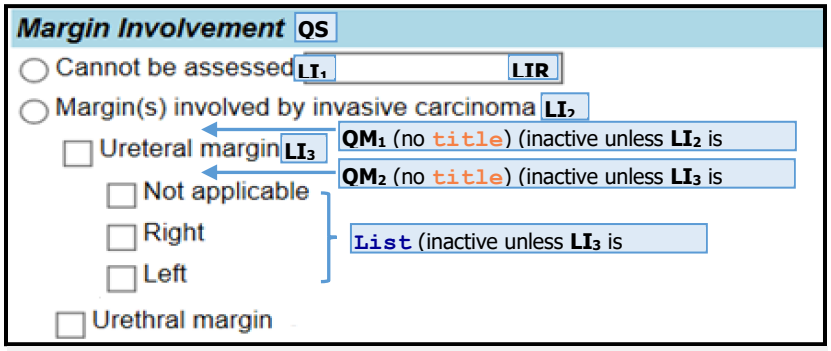
In Example 32B, **QS2** is a direct child of **QS1**. In other words, **QS2** is not a child of LI.1b. Thus, **QS2** is activated when any **ListItem** (LI.1a or LI.1b) is selected from **QS1**.

The SDC model works well when IA is implemented in the DEF using appropriate enabled/disabled or visible/invisible properties. If a parent DEF control is deactivated, then that all descendant controls must also be deactivated. If a user enters responses in a **Question**, and that **Question** becomes deactivated due to activity elsewhere in the DEF, then the responses for deactivated Q should not be present in the stored dataset.

[58] Some SDC implementations permit descendant **ListItem**s to be selected (e.g., LI.2a) before ancestor **ListItem**s (LI.1b) are selected. This requires that descendant **Question**s (**QS2**) are activated in the DEF before the user responds to ancestor **Question**s (**QS1**). In these cases, the ancestor **ListItem** (LI.1b) must be automatically-selected when a descendant **ListItem** (LI.2a or LI.2b) is selected by the user. Note that ancestor QR and LIR responses (e.g., **QR3**) cannot be automatically filled out when descendants (LI.4a or LI.4b) are selected. There are other issues with this model as well. The programming for this approach is more difficult and error-prone and can result in inadvertent errors from improperly completed DEFs. Further discussion of this model is beyond the scope of this document.

7.2.1 Implied Activation with Complex Nesting and Untitled Sub-`Question`s

In the following examples, observe that `Question`s **QM₁** and **QM₂** are nested under parent `ListItem`s (**LI₂** and **LI₃** respectively). QM₁ and QM₂ only become activated[59] if their parent LIs are selected. This is an example of IA, described above. When rendered as form controls, the child XFCs (and all their descendants) are inactive/disabled until the parent `Question` response is *captured*. A QR response is captured by entering a response (fill-in) value in the DEF. A QS or QM `ListItem` selection is captured by selecting one or more LIs. Example 34 depicts a *complex* QAS (a QAS with descendant `Question`s) showing multi-level control nesting that mirrors the FDF XML form component nesting.



Example 34: DEF with Complex Nesting and IA (see XML in next example)

```
<Question order="306" ID="27461.100004300" title="Margin Involvement">                    QS
  <ListField>
    <List>
      <ListItem order="309" ID="14214.100004300" title="Cannot be assessed">             LI1
        <ListItemResponseField name="CBA_310" order="310">
          <Response><string name="CBA_312" order="312" maxLength="4000" /></Response>    LIR
        </ListItemResponseField>
      </ListItem>
      <ListItem ID="14221.100004300" title="Margin(s) involved by invasive carcinoma">  LI2
        <ChildItems>
          <Question ID="38871.100004300">                                               QM1
            <ListField maxSelections="0">
              <List>
                <ListItem order="318" ID="27175.100004300" title="Ureteral margin">     LI3
                  <ChildItems>
                    <Question order="320" ID="38915.100004300">                         QM2
                      <ListField maxSelections="0">
                        <List>
                          <ListItem ID="39185.100004300" title="Not applicable" selectionDeselectsSiblings="true" />
                          <ListItem ID="38916.100004300" title="Right" />
                          <ListItem ID="38917.100004300" title="Left" />
                        </List
                      </ListField>
                      <ListItem ID="38918.100004300" title="Urethral margin" />
                    </Question>
                    ... (closing tags)
```

List

Example 33: FDF XML with Complex Nesting and IA

Visibility of inactive controls is subject to personal preference and usability considerations. In general, the visibility of inactive controls is left to the FF design and usability team (i.e., the user experience or "UX" team) to decide.

Note that **QM₁** and **QM₂** have no `title` value in the XML, but the `ListItem`s do have `title` values. These `Question`s with no `title` value are often called *untitled* `Question`s. Although no **QM₁** or **QM₂** `Question` text is

[59] *Active* means that a form control can be manipulated or answered by the user. Thus, an activated control is both *enabled* and *visible* to the user. An *inactive* control is *disabled* (grayed out) and may also be *invisible*.

visible in the DEF, the nested **ListItem**s provide a hint to the existence of the 2 untitled **Question**s, and the **Question**s' meaning can be readily inferred from the QAS context. See section The Untitled Question for more information on untitled **Question**s.

In most cases, inactive or **readOnly** status should be rendered as grayed-out disabled controls, and not hidden. Consider that hiding and unhiding controls [changing visibility] can be distracting to users, so activation/deactivation behavior must be carefully evaluated.

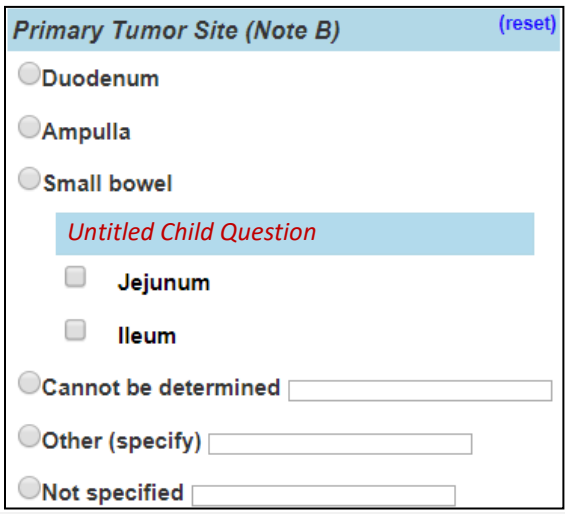
7.2.2 Complex Item Dependencies

Although IA covers most simple dependencies in SDC, some use cases may require that **Question**s become activated or deactivated based upon more complex criteria. For example, a hypothetical DEF may have a **Question** 4 that becomes activated when both **Question** 1 and **Question** 2 contain specific values, and **Question** 3 does not contain either of two specified values. These relatively complex cases are largely out of scope for the current document.

7.3 The Untitled Question

As noted above, missing **title** values in a **Question** results in the creation of "untitled **Question**s," where **ListItem**s are visible, but the parent **Question** text is not. Blank **title** text is used to unclutter the DEF when the meaning of "sub-answers" is obvious from the context, and **Question**s **title** text would be distracting. However, several some implementers have requested that the FDF suggest *alternative text* that reflects what the **Question** text could be, if it were displayed in the DEF. Therefore, the FDF files can supply these inferred terms as a **Property** with **propName=** "altText" for untitled **Question**s. However, the altText **Property** values may have a deleterious effect when the FDF authors have decided that **Question**s **title** content is best omitted. Importantly, altText **Property** values are not necessarily appropriate for use in synoptic report output. In general, reportText **Property** values, *not* altText **Property** values, should be used to customize report output.

In some cases, the altText **Property** value merely replicates the **title** content from the parent **Question**, because the parent **Question** text continues to be appropriate for the child **ListItem**s. For example:



Example 35: Untitled Question

In this case, the altText **Property** value for the *Untitled Child Question* may be assigned the same text value as the parent **Question** (**title** = "Primary Tumor Site (Note B)"), or it may be assigned a more specific descriptive value, such as "*Small Bowel Site*." As a general guide, altText **Property** values from untitled **Question**s should be suppressed whenever it merely duplicates the parent **Question**, as in the above example.

The altText **Property** values should be created in a terse manner consistent with the original **Question** semantics. However, this is not possible in some cases, especially when terse wording can result in ambiguous terms.

The altText **Property** value for the untitled **Question**s can be found as a **Property** subsumed under the untitled **Question** element. Implementers should check with their end-users to determine whether they wish to use the supplied altText **Property** value, substitute their own appropriate text, or suppress the DEF **Question** text entirely.

We recommend that altText **Property** values be used only for internal purposes (e.g., as a guide to the visual identification of **Question**s in database queries) and not be displayed as visible text in the DEF unless approved by the end-users.

7.4 The mustImplement Attribute (mI)

The **mustImplement** attribute may be found on any XFC, except **InjectForm**. By default, an XFC must be implemented on a form, so that users can capture a response in that control. If **mustImplement** = "True" (the default) or if **mustImplement** is missing in the XML, then the XFC must be implemented in the form.

However, if **mustImplement** = "False", then the XFC (and all descendants) need not be implemented as a control in the DEF.

7.4.1 Optional **ListItem**s

In rare cases, individual **ListItem**s to a required **Question** can be optional to implement (display). For all optional **ListItem**s, the value of **mustImplement** is "False" on the **ListItem** element. In some use cases, a plus sign (+) prefix may be used in the **ListItem**'s title value in the SDC XML, and the "+" may also appear in the DEF to indicate this optional status. Display of "+" signs in DEFs is subject to user preference.

7.5 Optional, Required and Conditionally-Required (CRQ) Responses

The appearance of a control in a DEF does not imply that it *must* receive a response from the user. Additional attributes are used to indicate if a response must be captured.

If **minCard** >0 for a **Question**, then the **Question**, if activated, must receive a response. If **minCard** >0 for a **Section**, then any required and activated **Question**s (with **minCard** >0) in the **Section** must receive a response. (i.e., the **Section** may not be skipped)

If **minCard="0"** on a **Question**, then the **Question** need not be answered. It is *optional to answer*. However, if it is answered, then any required (**minCard=** "1") and activated child **Question**s must also be answered.

If **minCard**="0" on a **Section**, then this **Section** may be skipped entirely, depending on user discretion. Its **Question** content is *optional to answer*. However, if the user begins to answer any **Question** in the **Section**), then all of the activated required **Question**s in the **Section** must be answered.

- The **Section** behaves as if it had a button at the top. If the user presses the hypothetical button (equivalent to answering any **Question** in the **Section**), then the **Section** changes to required (**minCard=** "1")

An additional situation can arise if a **Section** or **Question** is marker with **mustImplement="True"** and **minCard="0"**. In this case, the XFC must be shown in the DEF (it is *required to implement*), but answering it is *conditional* on some external factor that can't be known by the FDF author/modeler. This pattern is called *conditionally required* (**CRQ**) – the *user will decide* whether the optional (**minCard="0"**) **Section** or **Question** is applicable, and if it is applicable, it is treated as required (**minCard=** "1"), as we described earlier. A common use for CRQ items is when using required DEFs, but only the user can know which parts of the DEF are applicable in the specific situation when the DEF is being filled out.

- For example, consider a CRQ **Section** (i.e., with **mustImplement="True"** and **minCard="0")** that contains 2 top-level required child **Question**s, and with a **Section** instruction that reads "Complete this section only if lymph nodes are present in the specimen". In this case, if the user sees that lymph nodes

are present (which, of course, is unknown to the FDF author and modeler), the DEF user must fill out the activated required **Question**s in the **Section**. If any single **Question** in the **Section** is answered (because lymph nodes are present), then the entire **Section** is considered "required" (as if it had `minCard="1"`) and thus the **Section**'s second child **Question** must be answered as well.

- CRQ **Questions** (or a DEF text instruction preceding them) should contain text with conditional words like "required only if," "if known," or similar language or intent. The ability to answer CRQ **Question**s may depend on the user's responses to other DEF **Question**s, and/or on information not explicitly present in DEF.
- See section 7.6.1 for another example and information on reporting results from CRQ **Question**s.

One additional attribute determines when fill-in responses must be captured in a selected LIR control:

- If `responseRequired="True"` for an LIR's **ListItemResponseField** element, then if that **ListItem** control is selected by the user, a valid value must be entered in the **ListItemResponseField** control (the "fill-in" area). The datatype is Boolean. The default is `False`.

In Example 36, the use of `mustImplement`, `minCard` and `responseRequired` are demonstrated.

```
<Question ID="32780.100004300" mustImplement="false" minCard="0">
  <ListField>
    <List>
      <ListItem ID="32781.100004300" title="Health Canada Approved (specify test / vendor)" mustImplement="false">
        <ListItemResponseField  responseRequired="true">
          <Response><string  maxLength="4000" /></Response>
          ... (closing tags)}
```

`mustImplement="False"` means that this **Question** need not appear in the DEF.

`minCard` = **"0"** means this **Question** does *not* need to be answered (it is optional)

`responseRequired="True"` means that, if this **ListItem** is selected, then it *must* receive a fill-in response from the user. This holds even if the **Question** is optional (as in this case).

Example 36: Required responses

An implementation *may* use DEF color-coding and/or other visual representations to indicate that a required item must have a response captured, and/or to indicate CRQ **Section**s and **Question**s.

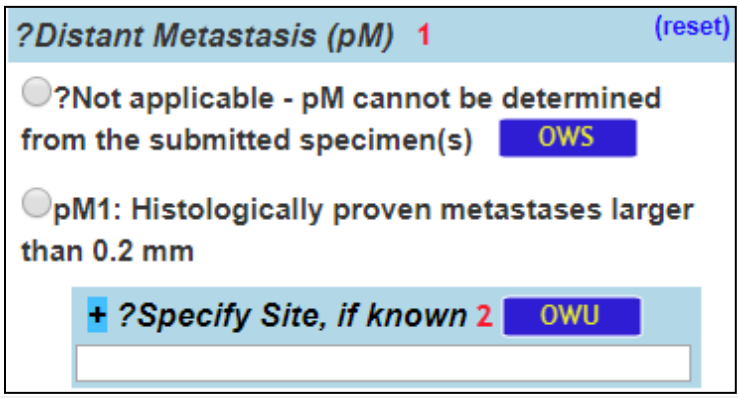
In some DEFs, optional **Section**s, **Question**s and **ListItem**s are prefixed with a plus (**+**) sign. In the SDC HTML reference DEFs, the "+" is used when `minCard="0"` on **Question**s, and when `mustImplement="False"` on **ListItem**s.

An entire FDF can be considered optional as well, and this will be determined for each use case, e.g., with accreditation or quality assurance requirements. An optional SDC FDF does not imply that every **Question** and **Section** will also be optional to answer (`minCard="0"`). Thus, XFCs in an optional FDF can still be flagged as required (`minCard > 0` and/or `mustImplement="True"`). Once a decision is made to use the optional FDF, the required parts must still be enforced by the DEF validation engine.

7.6 Conditionally Reported (CRP) Behaviors and Reporting from the DEF

All XFCs that have `mustImplement="True"` must appear in DEF so that users may complete them. However, the *responses* from some of these XFCs should be **omitted** from **reports** under specific circumstances. The selective omission of user responses from the report is called *conditional reporting* (CRP), because the inclusion of the data in a report is conditional on a few kinds of **Question** and **ListItem** metadata.

The two basic types of CRP items (OWU and OWS) are described next.



Example 37: Conditionally Reported (CRP) behaviors

7.6.1 The Omit When Unanswered (OWU) Model

As a default rule, any unanswered **Question** should be omitted from the DEF's report. Similarly, any **Section** with no answered child **Question**s should be entirely omitted from the DEF's report, by default. This is a general rule that applies regardless of the **mustImplement** and **minCard** content.

A special case occurs with CRQ **Section**s and **Question**s. The metadata to describe this case consists of **mustImplement** = "True" and **minCard** = "0". This means that a **Question** or **Section** must be displayed on the DEF, and it *must be answered* if it is applicable according to the specified condition (e.g., "if known").[60] However, the presence of the specified condition can only be determined by the DEF-user. This situation is demonstrated with **Question** 2 in the above figure. In this special case, the DEF display style shown in Example 37 includes a "**?**" symbol in the **Question** **title** to indicate that the **Question** has CRQ status. It also includes a **+** symbol to indicate **minCard** = "0". This special CRQ case is called *Omit When Unanswered* (OWU). Unanswered CRQ **Question**s and **Section**s are not reported, just like all unanswered **Question**s and **Section**s.

7.6.2 Omit When Selected (OWS)

If a **ListItem** has **omitWhenSelected** = "True" (**Question** 1 is the above figure), then selection of that **ListItem** should result in the default omission of the entire QAS (and all descendants of the **Question** and all contained **ListItem**s) from the *report*.

Even when omitted from the report, OWS responses must nevertheless be validated normally by the DEF software and their responses should be saved in the SDC data store (e.g., EHR database), as with any other answered **Question**. Storage of the non-reported data may be needed for several purposes such as quality or medico-legal review.

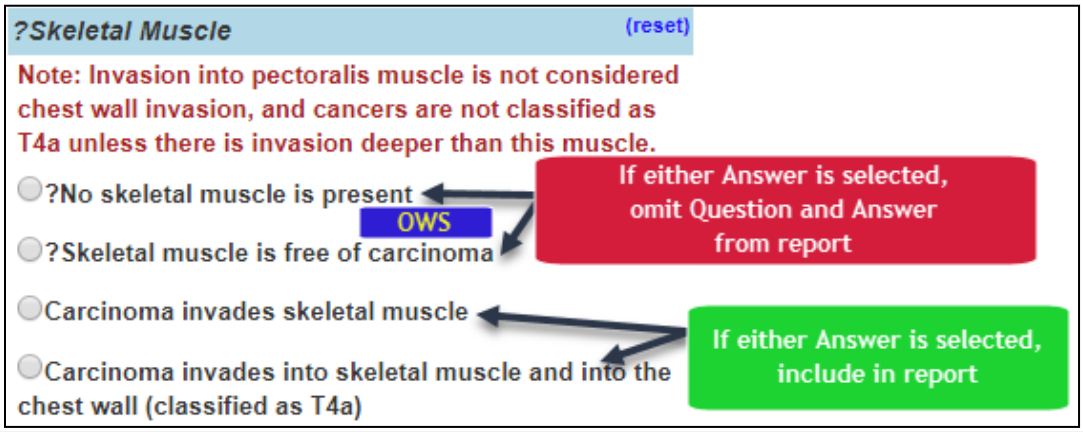
If the "?" display model is used for CRQ, then the **Question** and all OWS-flagged **ListItem**s should have the "?" prefix with the **title** attribute. The "?" on the OWS-flagged **ListItem** is simply a convenient visual flag for **omitWhenSelected**="true" in the SDC XML, and it informs the DEF-user that, if the OWS **ListItem** is selected, the QAS will be omitted from the report.

The *report-generator* must recognize the **omitWhenSelected** metadata, and, by default, omit the appropriate XFCs from the report. The report generator must also allow users to override the OWS default behavior if desired, thereby displaying OWS **Question** and **ListItem** responses in the report.

Unlike OWU **Question**s, *OWS* **Question**s *must be answered*, even if they are omitted from the report. Thus, unanswered OWS **Question**s should be flagged by the DEF validation mechanism to unobtrusively warn the user.

[60] Note that **minCard** = "0" will also cause a **+** prefix to be displayed on all OWU **Question**s in the SDC HTML (see **Question** 2 in the figure), and this **+** is a signal to users that the **Question** may be left blank if inapplicable. Despite the **+** prefix, all CRQ **Question**s *must be displayed* in the DEF, because in OWU cases, **mustImplement** = "True", and they *must be answered* if the DEF user determines that they are applicable.

However, not answering an OWS **Question** should not affect accreditation status. This is because accreditation reviewers look at reports, not at the stored answers. Thus, from the report alone, it is not always possible to tell the difference between an unanswered OWS **Question**, and one that was answered but properly omitted from the report. The limitations of OWS are recognized by the DEF authors, but this validation and reporting technique is an intentional approach to producing less verbose reports.



*Example 38: Two OWS **ListItem**s*

In the above example, two **ListItem**s (red box) are marked as OWS in the XML. When either one is selected, the **Question** and its responses (the selected **ListItem**s) are omitted from the report, by default.

7.6.3 Use of the "?" Prefix Display Model

SDC Implementers should use the FDF metadata to determine the presence of CRQ and CRP items. Implementers should choose an appropriate visual method to flag the required, optional, CRQ and CRP **Question**s, LIRs and **Section**s displayed in a DEF.

In one DEF convention, the `title` values of CRQ and CRP Items are prefixed in the DEF with the "**?**" symbol in the FDF (see above figure), so that an end-user may recognize when and how an item will be conditionally omitted from the report. Use of the "?" symbol in the DEF is not required, but alternate visual cues should be approved by the end-users. In the "?" display model, all CRQ **Question**s and all OWS **ListItem**s and their **Question**s have a "?" prefix.

?" prefixes should never be displayed in SDC reports. When "?" is used, a `reportText` **Property** with the corrected report text should be provided in the FDF metadata for each affected XFC.

7.7 Contiguity of **ListItem** Lists

The *answer list* resides inside the **Question/ListField/List** element tags. Within a set of **ListItem** elements, no **Section**, **Question**, **ButtonAction** or **InjectForm** may appear. (However, these items may appear after the **Question** element is closed, or nested under any **ListItem**.) In contrast, a "List Note**"** is a **DisplayedItem** XFC that can appear inside a **ListField/List**, in any position.

The following example overlays another color scheme to highlight a few points. Green elements are legal, and *red italic* elements are illegal according to the SDC Schema. The example shows the **List** element tag which wraps the answer list for a Question. The **List** tag contains two **ListItem**s and one **DisplayedItem**. The ***Section*** and ***Question*** elements are inside **List** and are thus illegal in the SDC XML. If *Section* and/or *Question* appeared immediately after the closing **</Question>** tag, they would be legal.

```
<Question>
  <ListField>
    <List>
      <ListItem/>
      <Section/>          (illegal)
      <DisplayedItem/>    (This is a legal "List Note")
      <ListItem/>
      <Question/>         (illegal)
      <ListItem>
    </List>
  </ListField>
</Question>
```

Example 39: Contiguity of `ListItem` Lists

In addition, most XFCs (all except LI) may appear indented under a legal **`ListItem`**, but they must first be wrapped in a **`ChildItems`** element, as shown in the following example. Although most XFCs can be legally nested under a **`ChildItems`** tag, a **`ListItem`** may only be nested in a **`Question/ListField/List/ListItem`** structure. The red **`ListItem`** and other tags are thus *illegal*:

```
…
<ChildItems>
 <Question>
   <ListField>
       <List>
          <ListItem>
             <ListItem/>             (illegal – not under <List>)
             <DisplayedItem/>        (illegal – not under <ChildItems> or <List>)
             <Section/>              (illegal – not under <ChildItems>)
             <Question/>             (illegal – not under <ChildItems>)
             <ChildItems>
                  <Section/>
                  <Question/>
                  <DisplayedItem/>
                  <ListItem/>         (illegal – not under <List>)
             </ChildItems>
             <ListItem/>             (illegal – not under <List>)
          </ListItem/>
          <DisplayedItem/>
          <ListItem/>
          <ListItem/>
       </List>
    </ListField
  </Question>
  <Section/>
  <Question/>
  <DisplayedItem/>
</ChildItems>
…
```

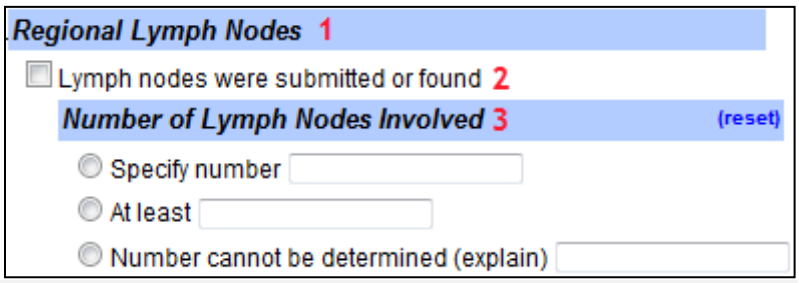
Example 40: Illegal XML structures in `List\ListItem`

7.8 The Null Check Box

The SDC model does not generally recommend the use of a 3-state (true, false, null) or "nullable" checkbox as a **`ListItem`** control, since all **`ListItem`**s/checkboxes must represent a value of `True` (selected) or `False` (unselected). **`Question`**s that require an explicit *Un*answered (Null) choice (as distinct from `False`) should preferably be reformulated as a QS **`Question`**, with mutually exclusive choices such as Present/Absent/Unknown/Cannot be Determined.

7.9 The Single Check Box

The figure below is an example of a **`Question`** (**1**) with a Single Check Box (**2**) and Sub-**`Question`** (**3**)



Example 41: The single check box

By SDC modeling convention, any **Question** with a single **ListItem** should be displayed as a QM. This is because a QM is typically implemented with checkboxes that can be readily deselected to undo a previous **ListItem** selection. On the other hand, a QS is usually implemented as a combo box or option button paradigm, and these paradigms do not typically support deselecting **ListItem**s.[61] As described earlier, the QM status is set on the parent Question, setting `maxSelections = "0"`.

In many cases, a QM with a single checkbox will be *required to answer* (i.e., `minCard` >0).[62] When validating template data for required **Question**s, this type of **Question** must be scored as "answered" even when the checkbox (**2** in the example above) is not selected, since unselected (unchecked) is a valid answer choice for a single check box.

7.9.1 Reporting from the Single Check Box

If the single checkbox (**2**) is not selected (checked), then neither the **Question** (**1**) nor the unchecked **ListItem** (**2**) should appear in the report. In normal cases, all Q1 descendants (**3**) would also be omitted from the report. This behavior (omission from the report) is expected, e.g., by form designers and accreditation agencies. However, the activation and reporting of the sub-**Question** (**3**) also depends on the `selectionDisablesChildren` (***SDAC)*** attribute on **ListItem** (**2**), which will be described later in detail.

7.9.2 The Locked (`readOnly`) QAS

Occasionally, a QAS may need to be locked, disallowing any editing of its **ListItem**s or response values by the end user. This technique is used in conjunction with the provision of default values, such as `selected = "True"` on LIs, or default responses inside a QR ResponseField or the **ListItemResponseField** on a LI. The locked status is expressed by setting the **Question**'s `readOnly` value to "`True`".[63] A locked QAS block may be included in an FDF to represent something that is always present, by definition, for a particular type of FDF, e.g., a body structure or specimen for which the FDF was created. These items are included in the FDF to facilitate searching of the stored data.

The locked, non-editable, information may be optionally displayed on the DEF and report, but end-users sometimes prefer that locked items are hidden. If it is displayed, it should not be represented in the DEF in the form of a selectable control (e.g., combo box). To facilitate a consistent approach to querying common data elements, the locked information should be stored in the SDC data store (e.g., database), along with the other **Question**/**ListItem** pairs.

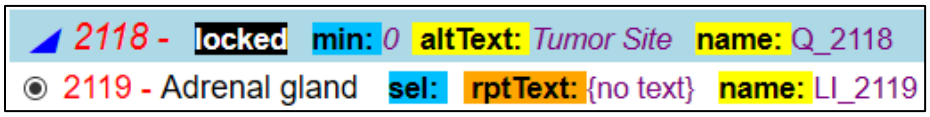
---

[61] There are ways around this "deselection" limitation of a QS, such as the use of a "reset" function as shown in the QS figures. However, most QM check box implementations support deselection natively.

[62] By writing "required to answer," we always assume that the QM is activated and reachable in the DEF's QAS tree hierarchy. The meaning of "required' depends on the use case, but in general, it means that the DEF will validate that the **Question** has a response and inform the user if a response is missing. That validation is impossible when there is only a single checkbox, because the computer can't distinguish an intentionally unchecked checkbox from a box that was accidentally skipped by the user.

[63] The `readOnly` status is not inherited by descendants of the locked **Question** or its **ListItem**s.

Locked **Question**s may have the value of **mustImplement** = "True", but with **minCard**="0" and a reportText **Property** **val** of "{No text}". In the Example 42, the **Question** **title** value is empty, and the altText **Property** value contains the hidden **Question** text (Tumor Site). This approach is used because in general, locked items are not intended to appear in the DEF or report. In rare cases, a locked element may be an essential piece of information that MUST display in the report. In these circumstances, the associated metadata (e.g., the rptText **Property**) will dictate the desired use and output. DEF designers may also use custom **Property** elements to embed required information into FDFs.



Example 42: The readonly (locked) attribute

Legend:
Locked **readOnly** = "True"
min: **minCard**
altText: altText **Property**
name: **name**
sel: **selected** = "True"
rptText: reportText **Property**

7.10 Flavors of Unanswerable (FOU)

A special group of **ListItem**s are useful when a **Question** cannot be definitively answered. Since these are similar to the well-known HL7 "Flavors of Null," they are called "Flavors of Unanswerable" (FOU). Some FOU **ListItem**s are represented as an LIR, allowing the user to type in a more detailed explanation of why the **Question** cannot be answered. The LIR responses for these are usually optional (**responseRequired** = "False"). A sample list of **title** content for FOU **ListItem**s follows:

- Cannot be assessed
- Cannot be determined
- Cannot be evaluated
- Indeterminate: this term, while still used rarely, has been deprecated from clinical forms because its meaning has been interpreted inconsistently as either "Cannot be determined" or "Equivocal"
- Inconclusive
- Not applicable: Not identified: used to replace more assertive terms like "Not present"
- Not known
- Not specified
- Unknown
- Unspecified

8 Rules

Sometimes the response captured by one QAS determines whether other non-child QAS items or Sections should be activated, or whether the response is used in a calculation of data value(s). These types of cases can be handled by SDC *Rules*, which are currently out of scope for this document. However, two such rules are currently in widespread use:

8.1 Selection Disables Children

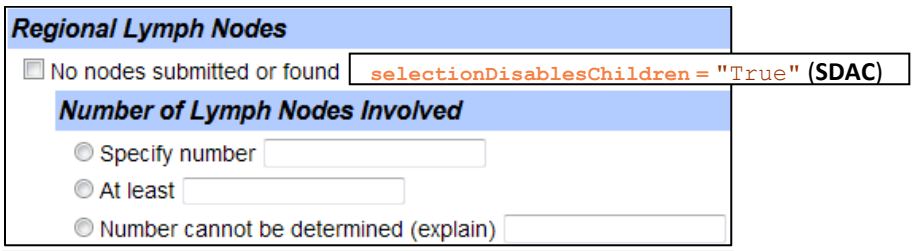
The `selectionDisablesChildren` **(SDAC[64])** attribute generally applies to a LI on a `Question` with only one LI.[65] Normally, the selection of a LI by the user will cause the first-level child controls to be activated. SDAC reverses this behavior – selection of the LI disables (deactivates) the child controls.

```
<Question ID="1867.100004300" title="Regional Lymph Nodes">
  <ListField maxSelections="0">
    <List>
      <ListItem ID="1868.100004300" title="No nodes submitted or found" selectionDisablesChildren="true">
        <ChildItems>
          <Question ID="42934.100004300" title="Number of Lymph Nodes Examined">
            <ListField>
              <List>
                <ListItem ID="1870.100004300" title="Specify number">
                  <ListItemResponseField name="Specify_286" responseRequired="true">
                    <Response><integer name="Specify_288" maxLength="3" /></Response>
                  </ListItemResponseField>
                </ListItem>
                <ListItem ID="54610.100004300" title="At least">
                  <ListItemResponseField responseRequired="true">
                    <Response><integer maxLength="3" /></Response>
                  </ListItemResponseField>
                </ListItem>
                <ListItem ID="1871.100004300" title="Number cannot be determined (explain)">
                  <ListItemResponseField name="CBD_295" responseRequired="true">
                    <Response><string name="CBD_297" maxLength="4000" /></Response>
                  </ListItemResponseField>
                </ListItem>
              </List>
            </ListField>
          </Question>
        </ChildItems>
      </ListItem>
    </List>
  </ListField>
</Question>
```

ListItem - SDAC

If the ListItem - SDAC control is selected, then this Question is deactivated

Example 44: `selectionDisablesChildren` (SDAC) in FDF XML



Example 43: `selectionDisablesChildren` (SDAC) in a DEF

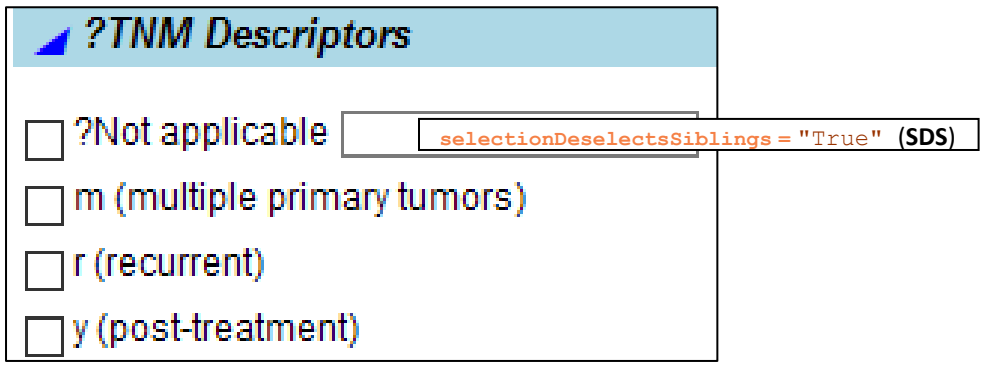
[64] SDAC was previously abbreviated as "SDC." The abbreviation was changed to avoid conflict with the ONC (Office of the National Coordinator) Structured Data Capture (SDC) standard.

[65] In these cases, the single LI is generally rendered as a check-box, not an option button, and the `Question` is treated as a QM. This is because a single checkbox can always be deselected, while a single option button cannot usually be deselected.

The reporting process for SDAC items are essentially the same as the process for reporting non-SDAC items. If an SDAC **ListItem** is unselected, the **ListItem** must not appear in the report. The parent **Question** **title** of the unselected SDAC **ListItem** is also omitted from the report, because unanswered **Question**s are never reported. If the SDAC **ListItem** is selected, the reported text of the SDAC **ListItem** and its parent **Question** is controlled by the **title**, reportText **Property** and **showInReport**, as usual. The activated descendant items of the *unselected* SDAC **ListItem** should appear in the report according to the usual reporting rules described for each item type.

8.2 Selection Deselects Siblings

The **selectionDeselectsSiblings** **(SDS)** attribute applies to any LI with a parent QM. If SDC is set to true on an LI, then selecting the LI will cause the DEF to de-select all selected sibling LIs and ignore any user-entered data entered in the previously-selected sibling LIs and their descendant captured responses. Deselecting the LI (the one with SDS = "true") permits the user to manually re-select the sibling LIs, and re-enable any descendant captured responses, if present.



*Example 45: **selectionDeselectsSiblings** (SDS) in a DEF*

```
<Question ID="15375.100004300" title="?TNM Descriptors">
 <Property propName="reportText" val="TNM Descriptors" />
 <ListField maxSelections="0">
  <List>
   <ListItem ID="2248.100004300" title="?Not applicable"
             omitWhenSelected="true" selectionDeselectsSiblings="true">
    <Property propName="reportText" val="Not applicable" />
    <ListItemResponseField>
     <Response>
      <string/>
     </Response>
    </ListItemResponseField>
   </ListItem>
   <ListItem ID="15377.100004300" title="m (multiple primary tumors)" />
   <ListItem ID="15379.100004300" title="r (recurrent)" />
   <ListItem ID="15476.100004300" title="y (post-treatment)" />
  </List>
 </ListField>
</Question>
```

ListItem-SDS

If the **ListItem-SDS** control is selected, then any selected sibling LIs are deselected

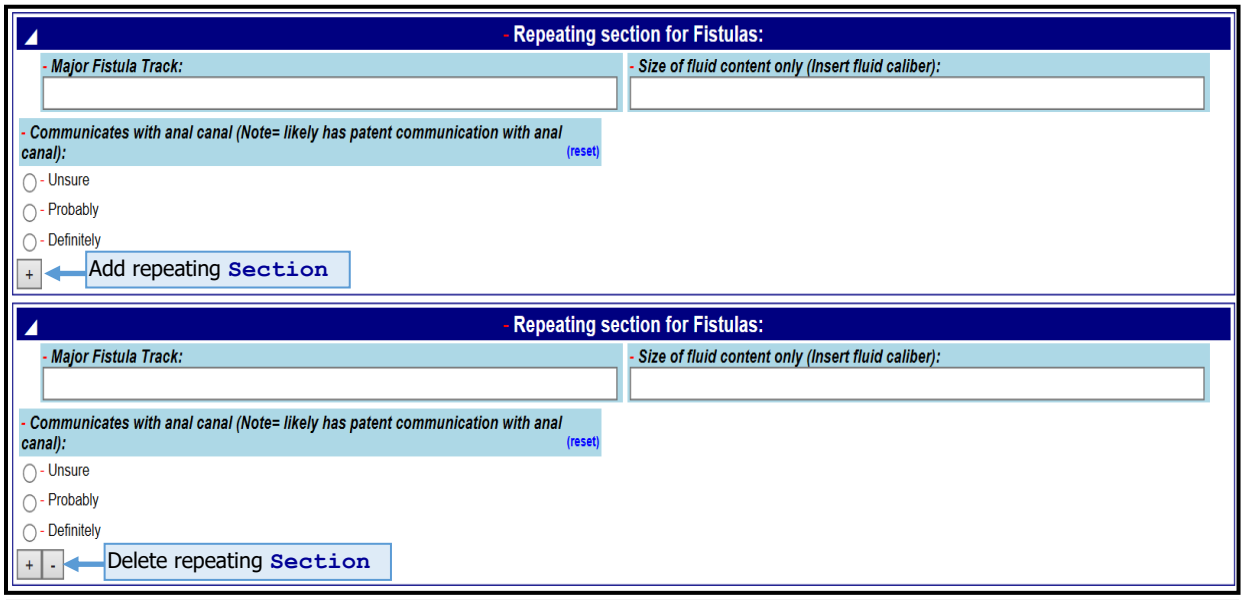
Example 46: selectionDeselectsSiblings (SDS) FDF XML

9 Repeating Sections and Questions

Section and **Question** components may be repeated any number of times in a DEF. The **maxCard** attribute specifies the maximum number of allowable repeats. If **maxCard** = "1" (the default) or is missing in the XML, the **Section** or **Question** is not repeatable. If **maxCard** = "0", then the number of repeats is unlimited.

If **minCard**="0", then the user is not required to capture a response(s) in the **Section** or **Question**. If **minCard** = "1" or greater, then the **Section** or **Question** must have its responses captured. If **minCard** >1, then the **Section** must be repeated at least the number of times in the **minCard** attribute (this pattern is rare).

Only one instance of a repeatable **Section** and **Question** component should appear in a DEF when it is first rendered. The user must be able to indicate when to display the next repeatable **Section** or **Question**. The FF must therefore include a clickable component (e.g., "**+**" and "**-**" buttons) or other interactive method with which the user can instruct the Form Filler to insert (or remove) a repeated instance of the **Section** or **Question**. This is illustrated in the following figure:



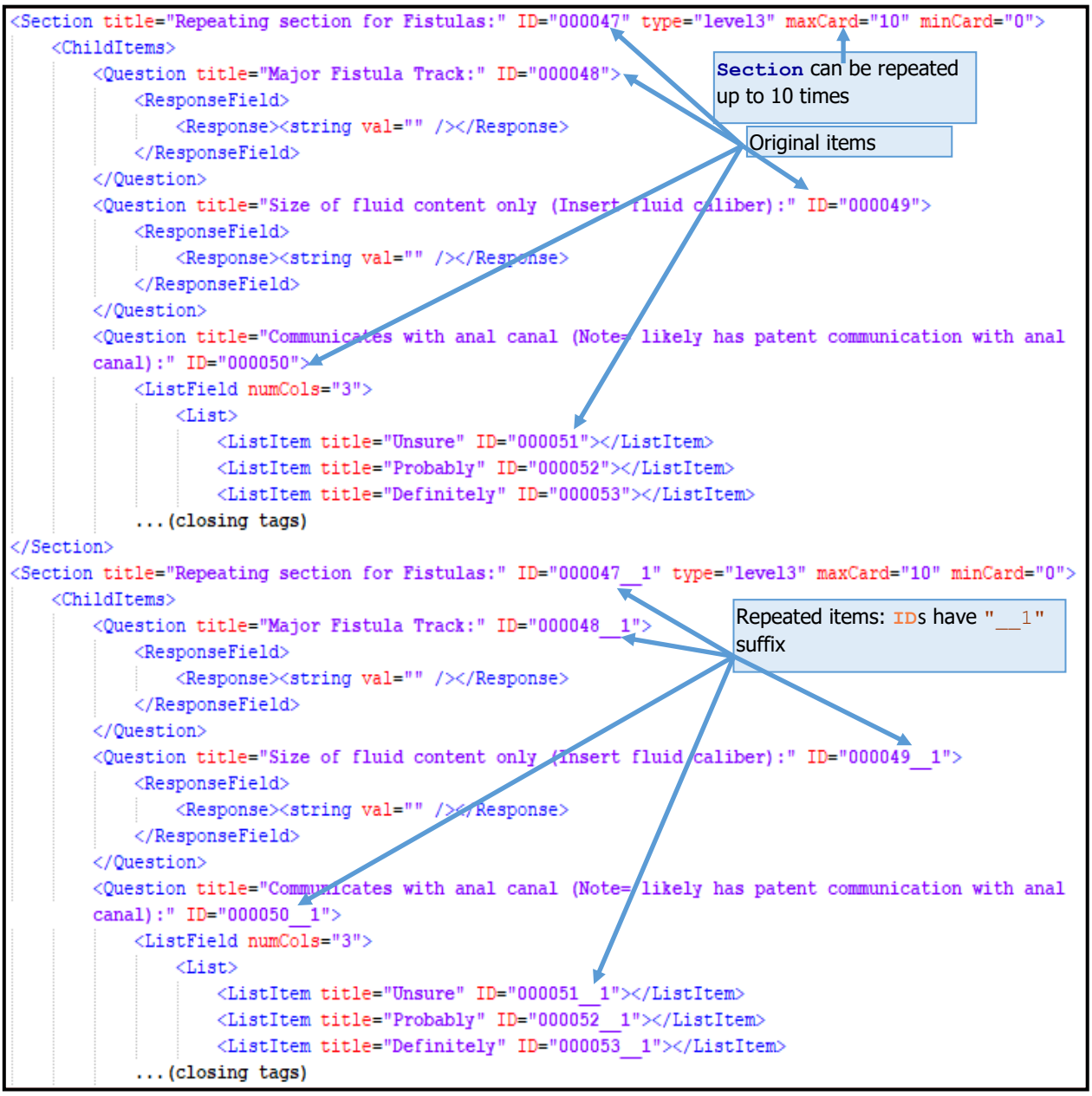
*Example 47: Repeating **Section**s and **Question**s*

9.1.1 Managing Repeated XFC and Component IDs with Monotonically Increasing Suffixes

When a **Section** or **Question** component is repeated, there must be a way to differentiate the **ID**s of each repeated DEF component. This is done by adding a numeric suffix to the **ID** and **name** (if present) of each repeated component/XFC and its descendant components, as illustrated in the next XML figure.

For each repeated component, an integer suffix (represented by "#" below) is created and appended to each **ID** in the entire repeated component **block** (i.e., the repeated component and all descendants). To manage the manipulation of **ID** attributes, The FF/DEF contains an **integer counter** function. The integer counter # starts at 0, and is increased by one every time a new repeated component block is added by the user, anywhere in the FDF/DEF. The XFC suffix begins with two underscores (__) followed by the shared counter integer (#) that identifies the repeated component block in the FDF. The same __# suffix is added to every **ID** and **name** in the repeated block. This same __# suffix replaces any previous __# suffix on any **ID** and **name** found anywhere in the repeated block.

If **order** is used in the FDF, then the repeated section should have incremented **order** content inserted by using small, sequential decimal increments to the decimal **order** content at the insertion point of the new repeated XML. It is preferable to leave all of the original **order** content unchanged.

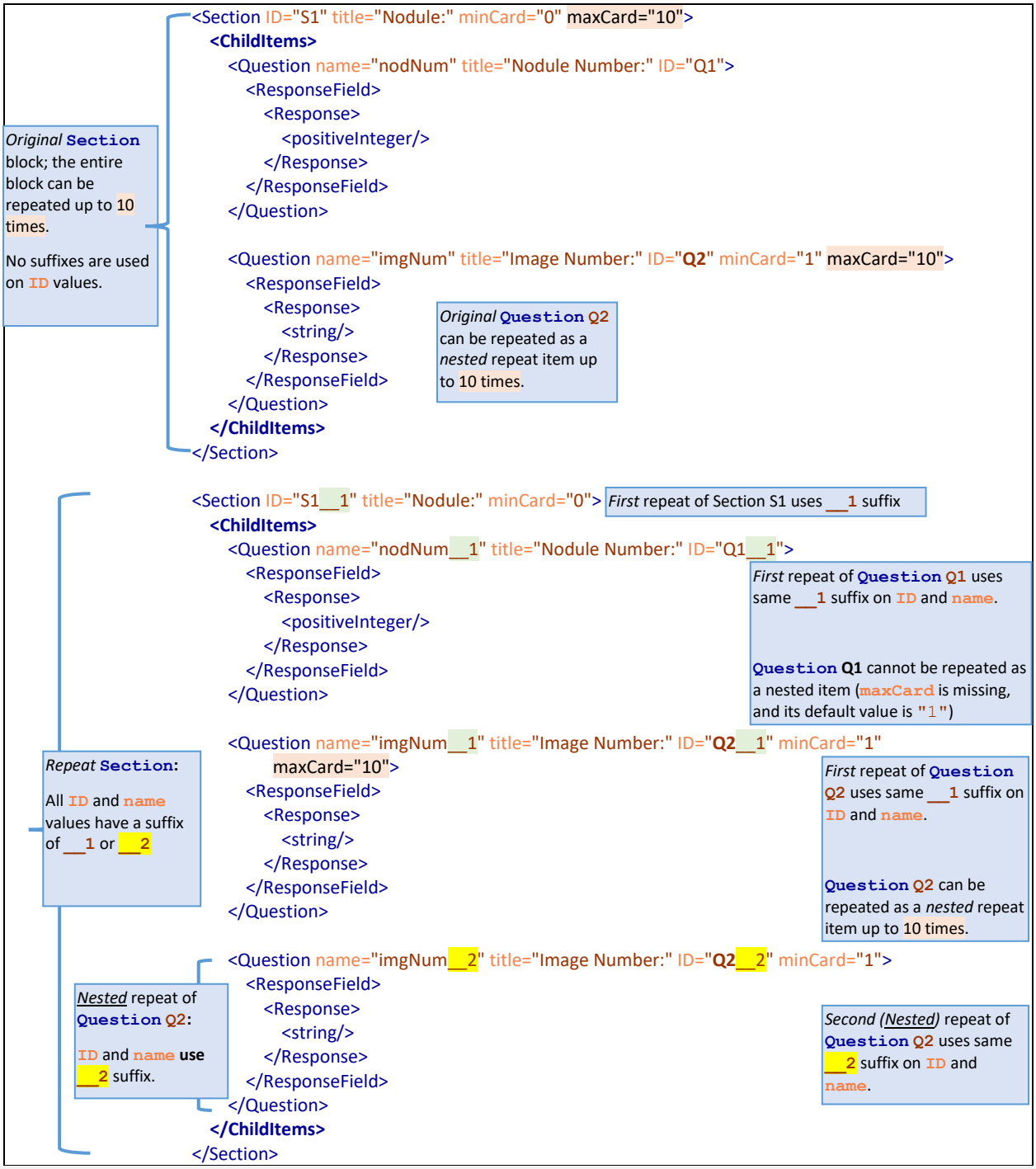



```
<Section title="Repeating section for Fistulas:" ID="000047" type="level3" maxCard="10" minCard="0">
    <ChildItems>
        <Question title="Major Fistula Track:" ID="000048">
            <ResponseField>
                <Response><string val="" /></Response>
            </ResponseField>
        </Question>
        <Question title="Size of fluid content only (Insert fluid caliber):" ID="000049">
            <ResponseField>
                <Response><string val="" /></Response>
            </ResponseField>
        </Question>
        <Question title="Communicates with anal canal (Note= likely has patent communication with anal
        canal):" ID="000050">
            <ListField numCols="3">
                <List>
                    <ListItem title="Unsure" ID="000051"></ListItem>
                    <ListItem title="Probably" ID="000052"></ListItem>
                    <ListItem title="Definitely" ID="000053"></ListItem>
            ...(closing tags)
</Section>
<Section title="Repeating section for Fistulas:" ID="000047__1" type="level3" maxCard="10" minCard="0">
    <ChildItems>
        <Question title="Major Fistula Track:" ID="000048__1">
            <ResponseField>
                <Response><string val="" /></Response>
            </ResponseField>
        </Question>
        <Question title="Size of fluid content only (Insert fluid caliber):" ID="000049__1">
            <ResponseField>
                <Response><string val="" /></Response>
            </ResponseField>
        </Question>
        <Question title="Communicates with anal canal (Note= likely has patent communication with anal
        canal):" ID="000050__1">
            <ListField numCols="3">
                <List>
                    <ListItem title="Unsure" ID="000051__1"></ListItem>
                    <ListItem title="Probably" ID="000052__1"></ListItem>
                    <ListItem title="Definitely" ID="000053__1"></ListItem>
            ...(closing tags)
```

Example 48: Managing repeated XFC and component IDs with monotonically increasing suffixes

9.1.2 Nested Repeats

As noted earlier, the repetition limit of an XFC block is defined by the `maxCard` content, highlighted in the example below. Nested repeats that occur within a repeated block of XFCs (i.e., repeated blocks inside repeated blocks) are treated the same way as the top of the repeated XML block: the integer counter is incremented again by one, and the original XFC `ID` is followed by `__#`, where # is the newly incremented integer shared by all `ID`s in the nested XFC block. The *original*, non-repeated XFC does *not* get a suffix on its `ID` value, but if it did, it would be "`__0`".



Example 49: Nested repeats

The first repeat of an XFC (the second instance of the XFC) gets a `__1` suffix on `ID` of all the XFCs in the repeated block. If a *nested* repeating XFC block (e.g., the `Question` with `ID` = `"Q2"`) repeats within the `__1` nested block, then the counter is incremented, in this case, to `__2`, and this new suffix is used for the internal nested block. In practice, if an XFC block already has an `ID` with a counter suffix, the counter is incremented for every `ID` in the additional repeated XFC block.

The optional `name` attribute is a unique identifier used to enable programmatic manipulation of captured data. If the `name` attribute is used, it receives the same suffix as the `ID` attribute.

10 DEF Validation and Reporting Results

10.1 Response Reporting Attributes:

When data is entered into an FDF by a user or by an automated mechanism such as auto-population, several kinds of required and optional tracking metadata are added to attributes in the FDF XML The metadata serve several general purposes:

- Add globally unique identifiers to each piece of FDF data; Creation of a child-->parent XFC (S, Q, LI, IF) tree to preserve the context of all captured data if they are stored in a non-FDF format. (applies to S, Q, LI, IF)
    - See FormDesign Instance Attributes: and `instanceGUID` and `parentGUID` in RepeatingType.
- Tracking of repeating FDF parts, keeping the various repeating parts grouped together in their original order. (applies to all XFCs)
    - See Repeating Sections and Questions
- Flag new and changed data (optional). (applies to **`FormDesign`**, S, Q, LI)
    - See FormDesign Instance Status Attributes**:**

10.2 Incomplete/Invalid DEF

A DEF is not considered complete (valid) unless responses to all applicable[66] required[67] QAS blocks are captured. Prior to submission, the user must be notified which required item (i.e., any **`Question`** with `minCard` > `0`, or any **`ListItemResponseField`** with `responseRequired = "True"`) does not have a suitable response captured. Even if required responses are missing, the FF, in most cases, should allow the user to save and submit the form.

10.3 Validating, Saving and Reporting from the DEF

DEF software must allow a user to save a DEF which has required QAS items that are unanswered.

Sometimes an attempt to save user responses in a DEF generates validation errors, e.g., missing responses to required QAS blocks. Although an "invalid" DEF may be incomplete or contain errors, it reflects the real-world situation, in which some data may not be available to the end-user, or the DEF may be improperly designed. Thus, SDC software must allow the end-user to save an incomplete template for later completion, review, editing and submission despite being incomplete.

DEF validation mechanisms should warn users when *applicable*[66] required QAS blocks are left unanswered or are answered in an illegal way (e.g., wrong data type). *Unanswered* refers to both missing selectable **`ListItem`**s as well as required fill-in values (LIR and QR). Applicable QAS blocks are those that can be reached in the QAS tree through the user's selection of **`ListItem`**s and entering appropriate responses in QR and LIR fields.

By default, unanswered/skipped QAS blocks should not be reported, even if the QAS blocks are required for accreditation. Local implementations may alter this default rule as they see fit.

---

[66] A QAS is only applicable if it is activated and has been reached by the user as s/he navigates through the form DEF hierarchy.

[67] i.e., `minCard = "1".` In the case of CRQ **`Question`**s and **`Section`**s, see Optional, Required and Conditionally-Required (CRQ) Responses.

11 Generating Reports from a DEF

Reports contain the user responses from one or more DEFs. Reporting user responses is often the primary reason for using a DEF. The desired report output can greatly affect the design and usability of a DEF. Satisfying accreditation requirements may depend on report layout. For example, CAP requires that reports are laid out in synoptic format.

Using the DEF format as a guide to design reports can result in suboptimal report readability. However, the optimal design of reports for each use case is a controversial issue. Some general considerations for report design follow. Most are subject to personal preference.

- Reports generally do not contain unanswered **Question**s or unselected **ListItem**s.
- In many cases, the **Question** or **ListItem** **title** text on the DEF should be altered to improve report readability.
    - As noted earlier, the DEF displayed text of each item is contained in the item's **title**. The **title** text should also be used when generating reports from the DEF, unless the text is overridden by the value in the item's reportText **Property**. If the **title** text should be completely suppressed in the report, then reportText **Property** will have a value of "{No text}" or **showInReport** = "False".
    - In some cases, changes to the **title** in the report can cause a misreading of the intended concept and can also cause an accreditation issue. Caution and testing is advised to ensure that the reportText value works well with all paths through the DEF tree.
- The layout of a report may not always follow the hierarchical layout of the DEF. In some cases, a left-aligned format may be preferred to a hierarchical layout.
- The separation of **Question**s from **ListItem**s can take several forms, e.g.,
    - Use of one column for **Question**s and another for **ListItem**s
    - Separation of **Question** and **ListItem** by punctuation such as ":" or underscores or a string of dashes or dots.
- The display of multiple **ListItem**s from a single multi-select **Question** can be accomplished with several formatting options. One popular format is to display comma-separated multiple **ListItem**s consecutively on the same line. However, complexities arise, especially when some **ListItem**s have fill-in values or sub-**Question**s. Other formats may report each **ListItem** on a separate line.
- The inclusion of **Section**s and "Report Notes" in reports is optional. However, context, readability and important information may be lost if they are omitted. If a **Section** has no content (e.g., none of its subsumed **Question**s are reported), then, by default, the **Section** should not be reported.
- Some **DisplayedItem**s are designed specifically for report output, and may not appear on the DEF.
- The ordering of **Question**s and **ListItem**s in reports may not be the same as the DEF. However, it is generally acceptable to use the DEF ordering and the **Section** structure, if desired.
- The use of rich text (e.g., rtf or HTML), table formats, alternately-shaded lines, images and color can improve the readability and acceptability of reports for some readers.
- DEF responses may be reported in more than one location in a single report or in multiple reports. For example, a short diagnostic section may include output from a few QAS responses. The same QAS responses may also be present in a more verbose section of the same report or in a separate report.

A complete treatment of report formatting is out of scope for this document. Some additional report guidance and examples can be found in the synoptic report examples available on the CAP website.

12 The SDCPackage and its Metadata

12.1 The SDC Package (*draft*)

```
<SDCPackage
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:ihe:qrph:sdc:2016 RFD+SDCRetrieveForm.xsd"
     xmlns="urn:ihe:qrph:sdc:2016"
     packageID="My_SDCPackage_ID">
   <XMLPackage>
     <DemogFormDesign
         baseURI="www.cap.org/eCC/SDC/IHE"
         ID="My_DemogForm_ID">
     ...[Form content goes here]
     </DemogFormDesign>

     <FormDesign
         baseURI="www.cap.org/eCC/SDC/IHE"
         ID="My_MainForm_ID">
     ...[Form content goes here]
     </FormDesign>
   </XMLPackage>
</SDCPackage>
```

Example 50: The SDC Package

FDFs are distributed in an XML wrapper called **SDCPackage** (**Pkg**). The **SDCPackage** can contain one or more FDFs along with metadata describing the contained FDFs and their usage. **SDCPackage** contains a child element called **XMLPackage**[68], and **XMLPackage** contains the FDFs. The **SDCPackage** element contains **packageID**, which uniquely identifies the set of packaged FDFs and related metadata in the **SDCPackage**. The FDF forms contained in the **XMLPackage** element (in the **DemogFormDesign** and **FormDesign** sub-elements) contain the **ID** attribute to identify each FDF form. The **ID** attribute may be used in conjunction with the **baseURI** and related attributes, described above for **FormDesign**.

The **XMLPackage** element, which is a sub-element of **SDCPackage**, also contains one or more **FormDesign** blocks, which hold the content from one or more FDFs. **XMLPackage** may also contain optional FDF demographic content in another FormDesignType element called **DemogFormDesign**, which if present, is placed before the **FormDesign** element in the XML layout. The **FormDesign** and **DemogFormDesign** each contain their own unique **ID** attribute. **DemogFormDesign** is used to contain a generic demographic FDF that can be reused in combination with multiple different domain-specific FDFs. **DemogFormDesign** is separated out as a separate FDF so that its content does not need to be copied into and maintained with the many domain-specific SDC forms that need to use the same kind of demographic information. In this manner, demographic content need only be maintained in a single **DemogFormDesign** FDF that suitable for one or more use cases in multiple **FormDesign** FDFs.

SDCPackage contains the following attributes:

- **packageID:** [dt: anyURI] This is the unique ID of the **SDCPackage**, which may contain more than one form. Required.
    - When combined with the baseURI, **baseURI** + **packageID** together form a composite globally-unique identifier (CGUI). To represent this concept, we create the **ID** content with the following formula:

[68] The **SDCPackage** may contain an **HTMLPackage** or a **FormURL** instead of an **XMLPackage**. However, this usage is out of scope at this time. It may additionally contain one **Admin** part, and one or more **SubmissionRule** and/or **ComplianceRule** parts. These are also out of scope for this document. Interested readers should consult the SDC Schema and the SDC Profile documentation.

lineage_version_sdcPkg.
In other words, we concatenate the components lineage, version and the text "sdcPkg" using "_" characters as separators. The suffix "sdcPkg" indicates that we are working with the Pkg variant, not the FDF, FDF-R, Pkg-R or the DE variant of SDC XML.

ID="My.Pkg.123_1.001.011.REL_sdcPkg",

- pkgTitle: [O] [def: ""] [dt: string] An optional human-readable label or description for the package. Optional.
- baseURI: [O] [def: ""] [dt: anyURI] This parameter is required in the SDCPackage element but is optional elsewhere. It identifies the organization that is responsible for designing and maintaining the SDCPackage contents. Equivalent to baseURI in FormDesign. Required.
- filename: [O] [def: ""] [dt: string] The filename to use when the current package instance is saved as a file.
  - For Pkg files without response data, one suggested format for filename is:

    lineage_version_sdcPkg.xml.
  - For assigning a filename to Pkg-R with instance documents, the following filename pattern may be used:

    lineage_version_instanceID_instanceVersion_sdcPkgR.xml.

    instanceID is a GUID that identifies an instance of a Pkg-R. instanceVersion is the version of the instance Pkg-R. However, this format can become rather long, and thus short GUIDs or other formats may be considered.
  - An alternative shorter format for a Pkg-R filename is:

    instanceID_instanceVersion_sdcPkgR.xml

    However, this format is less human-readable than the longer format that includes the lineage and version information.
  - FDF-R files should not reuse the filename of the empty Pkg. It is better to delete the attribute in the FDF-R than to reuse the FDF's filename value.
  - The filename datatype is string. It is not required.
- basedOnURI: [O] [def: ""] [dt: anyURI] URI used to identify the empty package "template" that that this package is based upon. In most cases, this should be a standard package that is modified and/or extended by the current package.
- lineage: [dt: string] A string identifier that is used to group multiple versions of a single package. The lineage is constant for all versions of a single kind of package. When appended to baseURI, it can be used to retrieve all versions of one particular package.
- version: [dt: string] A string that contains the version text for the current package. It is designed to be used in conjunction with baseURI and lineage.
- fullURI: [dt: anyURI] The full URI that uniquely identifies the current empty package template.
  - The URI is patterned after the SDC web service API. It is created by building up a REST-style query string from several components: baseURI, lineage, version , and doctype , using a format of component=value pairs, as in the following example:

    fullURI="_baseURI=cap.org&_lineage=My.Pkg.123&_version=1.001.011.RC1&docType=sdcPkg".

Note that each bolded component name is preceded by an underscore "**_**" and that component names and values derive from a **SDCPackage** attribute. Note that all "**&**" (ampersand) symbols are escaped using the standard XML/HTML "&" escape notation. The specific order of components shown in the URI is not required, but the displayed order is suggested for consistency and readability.

The only **fullURI** component not present as an attribute in **SDCPackage** is **docType**, which is always set to "sdcPkg" for Pkg, and is set to "sdcPkgR" for Pkg-R documents, which contain user data. For other SDC document types, **doctype** is set to "sdcDE" for DEs (including CDEs in registries), "sdcFDF" for SDC FDFs, "sdcFDFR" for FDF-Rs and "sdcMap" for SDC maps.

Note that the FM endpoint URI is not present in **fullURI**. This information may be found in the Pkg, under the **SDCPackage/Admin/RegistryData** element.

- **instanceID**: [FM or FF] [O] [def: ""] [dt: string] Unique string used to identify a unique instance of a form. Used for tracking form responses across time and across multiple episodes of editing by end-users. This string does not change for each edit session of a package instance.
- **instanceVersion**: [FF] [O] [def: ""] [dt: dateTime] Timestamp used to identify a unique instance of a package. Used for tracking form responses across time and across multiple episodes of editing by end-users. This field must change for each edit session of a form instance.
- **instanceVersionURI**: [FF] [O] [def: ""] [dt: anyURI] Globally-unique URI used to identify a unique instance of a Pkg with saved FDF-R responses. It is used for tracking Pkg responses across time and across multiple episodes of editing by end-users. The **instanceVersionURI** must change for each edit/save session of a Pkg instance (defined by **instanceVersion**), which includes any change in any of the FDF-Rs or ancillary data, files and metadata contained in the Pkg.

  The **instanceVersionURI** should be formatted similarly to the **fullURI** but must include values for **instanceID** and **instanceVersion**. The **instanceVersion** value is the release date/time for the new version, in W3C datetime format. An example **instanceVersionURI** is:

  - **instanceVersionURI**="**_baseURI**=cap.org&**_lineage**=My.Pkg.227&**_version**=1.001.011.RC1 &**_instanceID**=Abc1dee2fg543&**_instanceVersion**=2019-07-16T19:20:30+01:00&**_docType**=sdcPkgR "

    It is possible to create a shorter URI without the **_baseURI, _lineage** and **_version** parameters, as long as the URI is able to globally and uniquely identify and retrieve the instance and version of the FDF-R that was transmitted:

  - **instanceVersionURI**="**_instanceID**=Abc1dee2fg543&**_instanceVersion**=2019-07-16T19:20:30+01:00&**_docType**= sdcPkgR "

    Note that the FR webservice endpoint URI is not provided in the **instanceVersionURI**. The FR endpoint and its security settings may be found at **SDCPackage/SubmissionRule**.

    The **docType** for **instanceVersionURI** is sdcPkgR for content containing one or more FDF-Rs; sdcPkgR should also be used when the Pkg contains an optional **DemogForm** FDF-R in addition to a single **FormDesign** FDF-R as content. The **docType** for a Pkg with multiple FDF-R components and/or other content is also sdcPkgR.

    The specific order of components shown in the URI examples is not required, but the component order shown above is suggested for consistency and readability. The **instanceVersionURI** is not required, and may only appear in a Pkg-R; it is not allowed in a Pkg.

- **instanceVersionPrev**: [FM or FF] [O] [def: ""] [dt: dateTime] Timestamp value to identify the immediate previous instance of a Pkg instance. Used for tracking form responses across time and across multiple episodes of editing by end-users. This field must change for each edit session of a form instance.

13 Deprecated Content

The term "deprecated" is used in programming jargon to identify items that will soon be removed from use, and will then be unsupported. For the SDC, "deprecated" means that an XFC item has been "taken out of service" or removed from active use in a template. Deprecated XFCs and other changes over time may be viewed in the CAP's SDC Comparison tool (*link*).

13.1 When are Items Deprecated?

Deprecation of **Question**s and **ListItem**s can have adverse consequences on the use of queries that reference deprecated items. Therefore, items are deprecated only when the benefits clearly outweigh the implementation and analysis difficulties.

Items may be deprecated for several reasons. These include: improving clinical "correctness," generally at the request of FDF authors; improving support for DE or terminology mapping; fixing of defective QAS modeling; and updating DEs to keep current with the latest subject matter domain changes. Any item may be deprecated when it is dropped from the source guideline. Items may also be deprecated due to associated changes in the guideline, or changes due to FDF remodeling.

In general, **Question**s and **ListItem**s are deprecated when keeping them is judged likely to adversely affect the semantics, context and/or integrity of database queries that analyze those items. Deprecation is avoided whenever the **Question** or **ListItem** can be reused without adversely affecting query and data integrity. Deprecation of **DisplayedItem**s and **Section**s is unlikely to significantly affect query integrity, unless it affects the context in which DEs are embedded. Thus, these items may be deprecated when the **Section**/**DisplayedItem** text semantics change significantly.

In the past, if semantics were not adversely affected, **Section**s, **DisplayedItem**s were occasionally changed into other types of items (including **Question**s and **ListItem**s) without being deprecated. This practice has been changed, and deprecation in now used in these situations.

Question deprecation occurs when:

- A single-select **Question** (QS) or multi-select **Question** (QM) changes to a fill-in **Question** (QR). This usually involves loss of **ListItem**s.
- QS changes to QM or QM changes to QS
- QR changes to QS or QM. This usually involves addition of **ListItem**s. This occurred multiple times with the new Margin measurement remodeling in this release.
- QS/QM/QR changes to **DisplayedItem** or **Section**.

Question context changes how the **Question** is answered.

- Significant addition or deletion of **ListItem** choices, when the change is likely to significantly alter the selection frequency for the original set of **ListItem**s.

Question deprecation may not occur when the following isolated changes occur:

- Addition/deletion of some "less-significant" **ListItem**s. These include **ListItem**s **title** content like "None identified," "Not applicable," "Other (specify)" or "Other histologic type not listed above (specify)" etc. These are changes which may not significantly affect the frequency at which the original **ListItem**s would be selected.
- When an LI item changes to LIR, or LIR changes to LI, but the LI item is not itself deprecated (see below for the LI deprecation rules).
- When in doubt about the change and its effect on data integrity, it is better to deprecate the Question.

ListItems are deprecated when the following isolated changes to the **ListItem** occur:

- For a given **ListItem** LI1, if more specific **ListItem**s (e.g., LI2 and LI3) are added to replace or overlap with LI1, and these new **ListItem** s allow more discrete querying, then LI1 should be deprecated.

- For a given **ListItem** LI1, if adding or dropping other specific **ListItem**s in the **ListItem** list (e.g., LI2 and LI3) would create the likelihood that LI1 will be selected with altered frequency, then LI1 should be deprecated and replaced.
- A change in the **ListItem** text causes a change in the semantics
- The **ListItem** units change
- LIR changes to QR
- NOTE: In all of the above cases, the **Question** should also be deprecated.

ListItem deprecation generally does **not** occur when the following isolated changes occur:

- If a **ListItem** (LI) changes to LIR, but the **Response** part is only a *general comment* (free text) field.
  - However, if the new **Response** part is a *specific answer* to a sub-**Question** contained in the **ListItem**'s `title` text, such as "`Specify type`", or "`Other histologic type`", then the semantics of the **ListItem** have changed, and the LI must be deprecated and replaced with an LIR. If the LI is deprecated, then its **Question** must also be deprecated.
- If LIR changes to LI, when the deleted Response part is only a *general comment* field, and not a *specific response*, as defined above.
  - However, if the removed **Response** part is a *specific answer* to a sub-**Question** contained in the **ListItem**'s **title** text, then the original LIR must be deprecated and replaced with an LI. If the LIR is deprecated, then its **Question** must also be deprecated.
- If the **Response** metadata (e.g., **responseRequired**) of an LIR is changed, and
  - The Response part is only a general comment (free text) field,

  --OR--
  - The metadata change does not affect the LIR semantics or interpretation. For example, a change in units (e.g., cm to mm) would *require* deprecation of the LIR, but a change of datatype from **decimal** to **integer** might not require deprecation if the additional decimal precision is not significant.
- Parent QS changes to QM
- Parent QM changes to QS
- When in doubt about the change and its effect on data integrity, it is better to deprecate the **ListItem** and its **Question**.

14 Versioning

14.1 FDF Versioning (TBD)

14.2 Package Versioning (TBD)

14.3 eCC Versioning

Each FDF filename contains the file **FormDesign** version. The **FormDesign** version (e.g., 002.000.011) is incremented to mark any changes to the FDF file since the previous release.

The incrementing of the eCC FDF version follows a defined pattern, called the "three-dot version format" (**TDVF**):

14.3.1 The eCC TDVF version Format

See section 4.4 for more information on **FormDesign** attributes that contribute to FDF version management.

In some cases, the FDF files may change, even though the semantic content is unchanged. This may occur, for example, when the XML schema changes, or when new FDF metadata has been released as new XFC attributes in an FDF file.

For the CAP eCC FDF files, the full format of the version is 123.456.789.1000043. Recently, the trailing decimal namespace suffix (".1000043") has been replaced with a release status suffix: "123.456.789.REL".

- Section 1 ("123") is incremented only for major changes in a CAP Cancer Protocol (CCP), such as a complete rewrite, or a major change in AJCC version.
- Section 2 ("456") is used for changes that result in changed XFC IDs in an active eCC FDF.

  XFC ID changes occur when **Question**s or **ListItem**s are added or retired ("deprecated") in a template. Examples include significant rewording of a **Question** or **ListItem**, addition of **Question**s or **ListItem**s (e.g., adding a **ListItem** for "Not applicable"), and deprecation (or "un-deprecation") of individual **Question**s or **ListItem**s. In rare cases, a change in **Section**s, **DisplayedItem**s or instructions can alter the way that a **Question** is answered, thereby changing the context and/or semantics of **Question**s and **ListItem**s. This type of change causes ambiguity in the meaning of the template items over time, requiring deprecation of the affected **Question**s or **ListItem**s (and their ID**s**), in conjunction with the creation of new **Question**s and/or **ListItem**s (with new ID**s**).
- Section 3 ("789") is used for minor changes to the eCC FDF content in an active eCC FDF. Increments in the first 2 digits ("78") indicate the minor content change (e.g., spelling errors, changes to the visible text that preserve item semantics, or changes to the position of items) or metadata change (e.g., a changed data type that does not alter the fill-in semantics). The last digit ("9") is used as the "new XML flag" to indicate that the XML release file has changed. This digit may be incremented even when there have been no content changes, e.g., when the XML schema has changed. In this case, all the previous digits ("123.456.78") will remain the same as the previous FDF version, and only the last digit will be incremented. Released content changes always force an FDF change. Therefore, the last digit ("9") is always set to "1" whenever there is a content change in segments 1 or 2.
- When changes occur in segment 1, all digits in segment 2 are set to zero and segment 3 is set to ".001".
- When changes occur in segment 2, the digits in segment 3 are set to ".001".
- Finally, a release status suffix is appended to the growing TDVF version to indicate the rough proximity to release for the updated FDF. Multiple changes may occur to the FDF without affecting the version, as long as the FDF has not reached its final state. However, the release status suffix should be updated with each non-final FDF version's release. Examples of the release status suffixes are CTP1 - CTP9, RC1 - RC9, etc. CTP is a "Community Technology Preview" released before the quality review has begun, and RC is a "Release Candidate" that has received some level of quality review, but may not be ready for final release. An example is "001.002.001.**RC3**". "REL" is the final release suffix, and would appear like the following example:

"001.002.001.**REL**". More suffixes may be added over time, to indicate evolving release workflow. In addition, the eCC's TDVF is subject to change in the future.

14.3.1.1 eCC FDF TDVF Examples:

- 00**2**.000.00**0**.REL: original version 2
- 002.000.00**1**.REL: **xml-only change** [**new xml flag**]. This occurs when an FDF XML changes, but does not significantly affect the semantics of a DE, as perceived by a DEF user and data analyst.
- 002.000.00**2**.REL: another **xml-only change** [**new xml flag**]
- 002.000.0**11**.REL: **minor** new or changed content [**+ new xml flag**]
- 002.000.0**21**.REL: **minor** new or changed content [**+ new xml flag**]
- 002.00**1**.00**1**.REL: **moderate** (segment 2) change [**+ new xml flag**]. This occurs with addition or deprecation of an XFC `ID`. The second segment is incremented by 1, and the third (last) segment is reset to ".001", regardless of concomitant minor changes (e.g., fixed typos)
- 00**3**.**000.000**.REL: major (segment 1) change. The first segment is incremented by 1, and the second and third segments are reset to ".000.000". This may occur with a completely new FDF `version`, which may have many or all new `ID`s (e.g., this occurs with the release of a heavily-revised CAP Cancer Protocol). It will always occur with a major change in a staging system (e.g., the updates in the AJCC 8th edition staging system).

14.3.2 Relationship of FDF `version`s to CAP Cancer Protocol (CCP) Versions

Introduced in Feb 2011, the versioning ID system for the CCPs is not intended to cover the same types of versioning issues as the eCC/SDC `version` system for FDF files. CCPs are versioned for content changes, whereas eCC FDF files may receive a new `version` value for content changes and/or technical changes. Because CCP version changes may or may not affect the case summary section, CCP version updates may or may not affect a later FDF release `version`. Thus, any given `version` of an FDF file could remain constant through several versions of the parent CCP document. Conversely, an FDF file may undergo several `version` changes while the parent CCP remains in the same version.

In general, information flows from the CCP to the eCC. In some cases, however, eCC-generated `version` updates may be adopted in future versions of the CCP. A description of the CCP version system is out of scope for this document, but is available on request from CAP.

14.4 eCC Implementation Using IDs for FDF Items `ID`s and FDFs:

To create an unambiguous system of referring to each XFC in each FDF, XFCs are assigned a unique line-item identifier. These identifiers (`ID`s or Ckeys) are generated at CAP by using a unique integer value.

However, it is possible for end-users to customize FDF documents, and assign their own integer values to their custom template items. If multiple form designers assign simple integer values, inevitably some FDFs will end up using the same integer keys for different customized template items. Since these keys may be used to communicate data between different sites, the use of simple integer keys would result in confused and corrupted data.

Therefore, the original CAP eCC metamodel employed a solution for this problem: We appended a site-specific "namespace" to the base integer of the `ID`. For each end-user institution, its unique namespace integer is appended, after a decimal point, to the base integer `ID`, as follows:

[Local Unique Integer] + decimal point + [CAP-Assigned Unique Namespace]

123456 + "." + 1000043 = 123456.1000043

This legacy system is still in use at the CAP, but the use of the CAP-assigned namespace has been deprecated. The newer URI-based namespace system is described in section 4.4.3. This legacy namespace system originated with SNOMED CT.

15 Data Storage, Terminologies, and Transmission

The basic building block of FDF files is the DE. The DE model follows the common practice of representing information as question-answer pairs. In the SDC world, this translates into storing a **Question** ID and a **ListItem** ID for each **Question** and **ListItem** in the filled-out DEF. Alternatively, a response value (answer) may be stored with the Question.

An additional level of global specificity and uniqueness in data storage is made possible through the use of the instanceGUID and parentGUID attributes which are assigned to answered **Question** and selected **ListItem**, as well as to **InjectForm** and **Section.** These attributes that are found in the FDF-R responses may be stored optionally in database tables along with the IDs. However, the examples shown below omit these attributes for simplicity of presentation.

Although it is possible to use SNOMED CT, LOINC or other semantic coding systems for the representation and storage of **ListItem** data, complete sets of these codes for each FDF are not always available or stable over time. Therefore, the ID is a convenient surrogate key for the **Question**s and **ListItem**s. IDs may be mapped, in separate XML mapping files, to the appropriate semantic codes, depending on code availability and end-user needs. Thus, IDs provide unique identifiers for lines in a template that can map to external coding systems such as CDEs, LOINC, SNOMED CT, and NAACCR data items.

To enable querying based on external coding systems, code tables for terminology systems should be stored separately from the ID-based DE data. External codes should be linked to the **Question**/**ListItem** pairs by using the ID as a "foreign key" in a mapping table for the external codes. (The eCC team provides these ID/external code mapping tables in XML format, so there is no need to maintain them on your own.)

Since many coding systems (e.g., SNOMED CT and ICD-O-3) introduce changes to the code values or the meaning of codes, storing external codes together with the IDs would require regular metadata maintenance, would risk the introduction of errors, and might result in the editing of an electronically-signed and locked patient record. In addition, external codes may not be available at the time of data storage. By using a mapping table, external codes can be simply referenced via the ID; updates to the mapping table can be as simple as importing an updated eCC mapping file.

For some applications, it may be necessary to transmit semantic (e.g., SNOMED CT, ICD-O-3), terminological (e.g., LOINC) or use-case-specific codes (e.g., NAACCR codes for cancer registries) *in addition* to SDC IDs.

15.1.1 Terminology (ICD-O-3 & SNOMED CT) Maps

FDF files may be released well before a full set of terminology and CDE mappings are available.

To provide a functional and interoperable implementation of the FDF, it is critical that SNOMED CT codes (i.e., Concept IDs), and codes from any other terminology, are ***not*** used as *unique identifiers* for FDF **Question**s and **ListItem**s. Instead, ID values should be stored for each **Question** and for each selected **ListItem** in the template.

15.1.2 Data Storage

When saving FDF-derived data, essential FDF and XFC Identifiers should be saved for each encounter or edit session. Links must be maintained between the various instance version identifiers as they change over time. These identifiers are associated with an FDF, but may also be associated with SDC Packages. Each encounter thus creates an encounter/instance record that is linked to the individual pieces of FDF-DE-derived data that are stored alongside it.

The encounter record generally hosts a *unique identifier*. In the context of storing an encounter record as SDC-based data, a composite identifier can be derived from package and FDF metadata, including the package instanceID + instanceVersion, and the FDF instanceID + instanceVersion. (i.e., a single encounter record identifier key can be associated with the composite attribute content.) The key is linked to each individual FDF-DE instance record that contains the **Question**, **ListItem** and/or fill-in response data. The key thus acts as the "Entity" key in an Entity-Attribute-Value (EAV) storage model. Other identifiers (e.g., patient identifiers) are also linked to this key. For simplicity of presentation, the examples shown below will omit this Entity key.

Storage of DE-derived Data

When storing the ListItem data for a QS Question, a single Question code (the "ID" of the Question item) and a single ListItem code (the "ID" of the ListItem) are stored in the database.

When storing ListItem data for a QM Question, a single Question code (the ID of the Question item) and a single ListItem code (the ID of the ListItem) are stored in a database record for each selected ListItem (checked box).

Why is it important to store the Question ID in addition to the ListItem ID? During FDF updates, sometimes Questions are deprecated to indicate a change to the Question's ListItems, but some or all of the original ListItems remain in the list (they are not deprecated). Replacement of the Question (and its ID) ensures that new queries based on the previous Question ID will not return ListItems from the new Question. This is intended to prevent misinterpretation of query data when the IDs and semantics associated with the ListField/List/ListItems change. In other words, queries based on the deprecated Question ID will not return results combined from the new Question ID, and this result is desirable because the new and deprecated Questions are based on different ListItem sets.

If the Question has not been answered, then neither the Question ID nor the ListItem ID should be stored. In the case of LIR items, the user-entered response content of the ListItem's Response section should be stored in addition to the ListItem's ID. Following this model will help to ensure that the stored data is interoperable with other SDC implementations.

In the examples below, all user-entered response content is shown in a single table column (field) of datatype string. However, storage of this data in datatype-specific fields (or even in separate tables) may be preferred.

15.1.3 Usage Examples

The examples below do not include Entity keys, local primary keys, other record/row identifiers, patient identifiers, terminologies, audit fields, or other local metadata, as these may vary between implementations.

Single-Select Questions (QS):

When storing or transmitting ListItem data for a QS Question, the ID of the Question and the ID of the ListItem should be stored or transmitted. For example, if the user selects the ListItem "LI1" from the QS Question "Q1," the data may be stored as follows:

Question ID	ListItem ID	Response*	Repetition Index*
[Q1 ID]	[LI1 ID]	NULL	NULL

* To be described below

Multi-Select Questions (QM):

When storing or transmitting multiple ListItems for a QM Question, each ListItem ID is used together with the associated Question ID. The Question ID thus repeats for each selected ListItem. For example, for a Question (Q1) with selected ListItem LI1, LI3, and LI10 (out of, say, 15 possible checkbox ListItems), the following pattern should be used:

Question ID	ListItem ID	Response*	Repetition Index*
[Q1 ID]	[LI1 ID]	NULL	NULL
[Q1 ID]	[LI3 ID]	NULL	NULL
[Q1 ID]	[LI10 ID]	NULL	NULL

* To be described below

Response Questions (QR):

In the case of QR responses, the Question ID should be stored as above, but the ListItem ID should be left blank. In addition, the text of the response data should be stored in a response field.

Question ID	ListItem ID	Response	Repetition Index*
[Q1 ID]	NULL	[Response]	NULL

* To be described below

ListItemResponse (LIR):

In some cases, the selection of a specific LI requires or allows that a Response section (e.g., text and numeric) be completed. This is known as a ListItem-Response (LIR). LIRs will require that both the LI ID *and* the Response text value be stored in the database, usually as part of the same database record, as shown here:

Question ID	ListItem ID	Response	Repetition Index*
[Q1 ID]	[LI1 ID]	[Response]	NULL

* To be described below

Repeating Sections and Questions:

Occasionally, a Question or Section may need to be repeated in the DEF, the FDF-R and in the database. The number of times that a Section or Question may repeat is determined by maxCard, described earlier. Storage of repeating responses is straightforward because incrementing IDs are generated for each repeated XFC (see Repeating Sections and Questions). For storing SDC data from repeating items, the IDs from the repeating items are stored in the same manner as any QS, QM or QR. Some implementations may wish to store the repeat value, which is optionally incremented for each repeat of each ID, as shown below:

Here is a simple example with repeated nested fill-in Questions (QR):

QRa: Specify Location of each Additional Margin _____

QRb: Distance of Tumor from Margin (mm) _____

The data for 3 repeats may be stored with repetition indices (0, 1, 2) as follows:

Q ID	ListItem ID*	Response	Repetition Index
[QRa IDa]	NULL	"3 o'clock, with suture at 12 o'clock"	0 or NULL
[QRb IDb]	NULL	"0.3"	0 or NULL
[QRa IDa_1]	NULL	"9 o'clock, with suture at 12 o'clock"	1
[QRb IDb_1]	NULL	"2.0"	1
[QRa IDa_2]	NULL	"Deep margin"	2
[QRb IDb_2]	NULL	"1"	2

*In this example, no ListItem (LI) IDs are available for storage, since QR items are used.

15.1.4 Storage of Additional Metadata

Storage of additional FDF and SDCPackage metadata can provide advantages for data analysis. Storage of the XFC order can aid in sorting the DE data in the original FDF order for reporting or reconstitution of the original FDF. Storage of instanceGUID and parentGUID can help to reconstruct the original FDF context hierarchy for reporting and analysis. It is possible to reconstitute this information by reference to the original FDF-R (which

should be saved as well), but discrete database storage of some FDF metadata may have advantages for some use cases. Storage of fixed FDF metadata such as `units` and validation criteria, etc., may be referenced as needed from the FDF, or parsed into table format for faster reference.

15.1.5 Data Transmission

SDC has tested two different FDF-R-based transmission models, one based on SOAP, and a newer one based on REST. These are currently out of scope for this document.

However, other transmission models can also work. NAACCR has created a very capable transmission model using HL7 2.51 format for cancer registry eCC data transmission. This is the eCC transmission standard at this time. Much more information can be found in the NAACCR Vol. V documentation.

16 Converting SDC FDFs into DEFs

16.1 Automatic DEF Generation

Although it is possible to create custom static DEFs (e.g., web pages, Windows forms) for each FDF, this is not required. The FDF files can also be used for metadata-driven creation of DEFs via "just-in-time" (JIT) DEF generation.

To demonstrate the use of JIT implementation, a sample XML→HTML DEF generator (using XSLT and CSS style sheets) is provided with each release, in the FDF folder. **The JIT-generated HTML DEFs are NOT intended to be functional SDC applications**; they are only guides to a possible implementation technique and do not include all SDC functionality.

The FDF documents may be viewed as HTML by dragging them into an XSLT-compliant web browser (e.g., Firefox), by generating XSLT-converted HTML files with a tool such as Oxygen or XML Spy, or by writing programming code to transform the FDF file into HTML with the provided XSLT program.

An example JIT-generated DEF is shown here. Representative XFCs and related metadata are labeled:

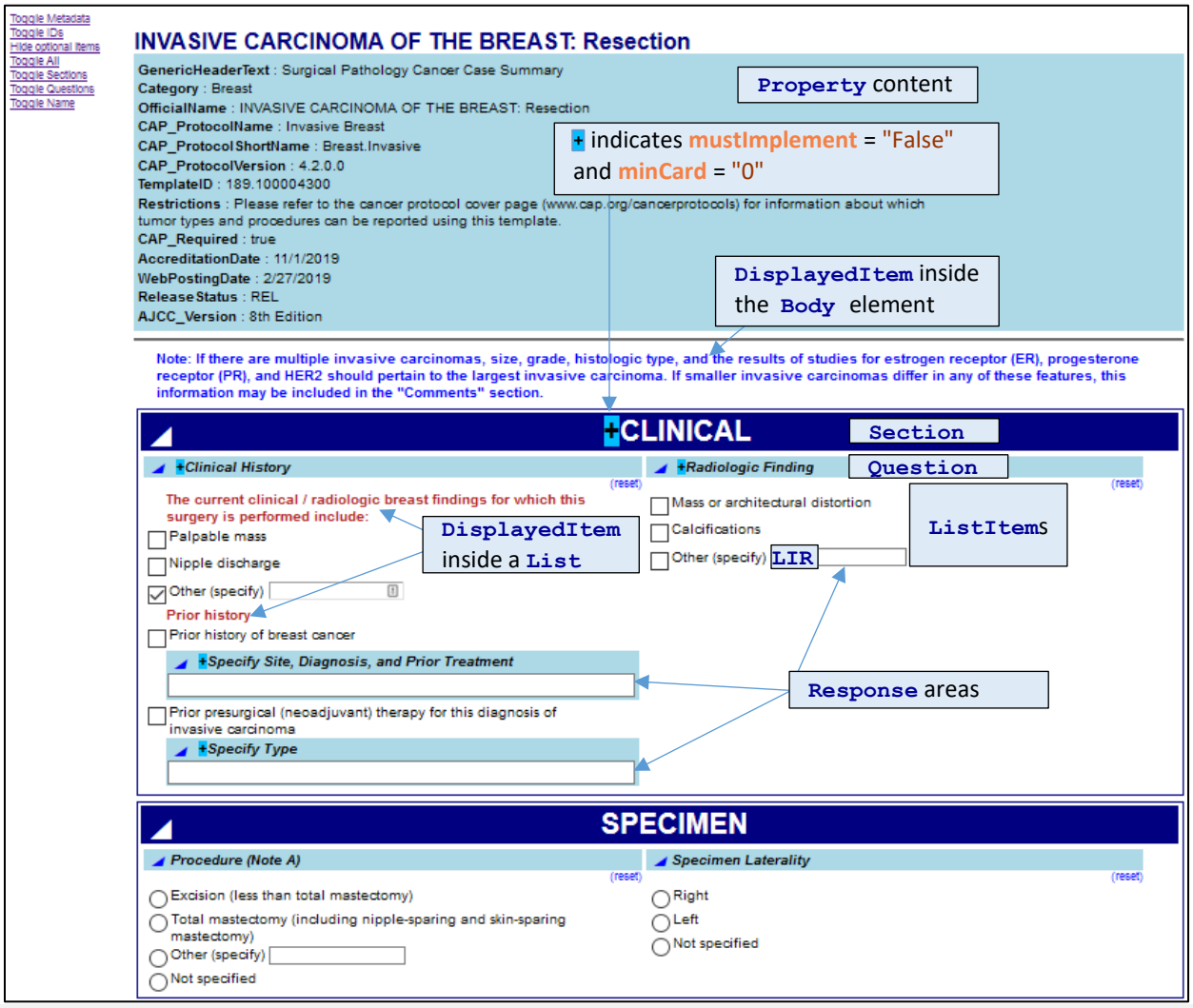


Figure 5: JIT-Generated HTML DEF

The eCC HTML pages contain toggle switches in the upper left corner for toggling the visibility of hidden metadata.

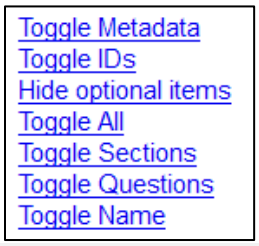


Figure 6: DEF Toggle Switches for Metadata

`Property` values subsumed by the `FormDesign` element are shown in the top of the DEF. Each bolded term is generated from the `Property`'s `propName` content, and the `Property`'s `val` content is displayed after the colon following each bolded `propName`:

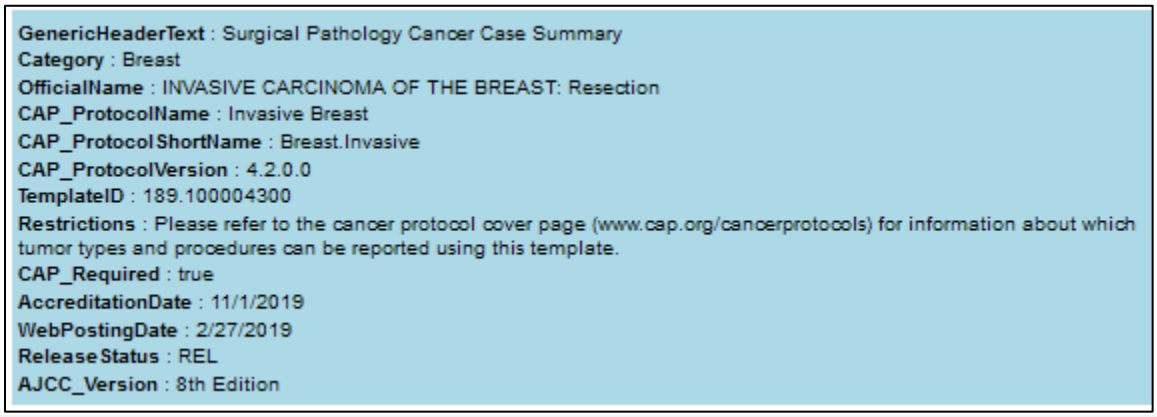


Figure 7: `FormDesign` Property Text in a DEF

`ID` content is shown in red, and other *hidden FDF metadata* is displayed with *colored icons* as shown in the following image:

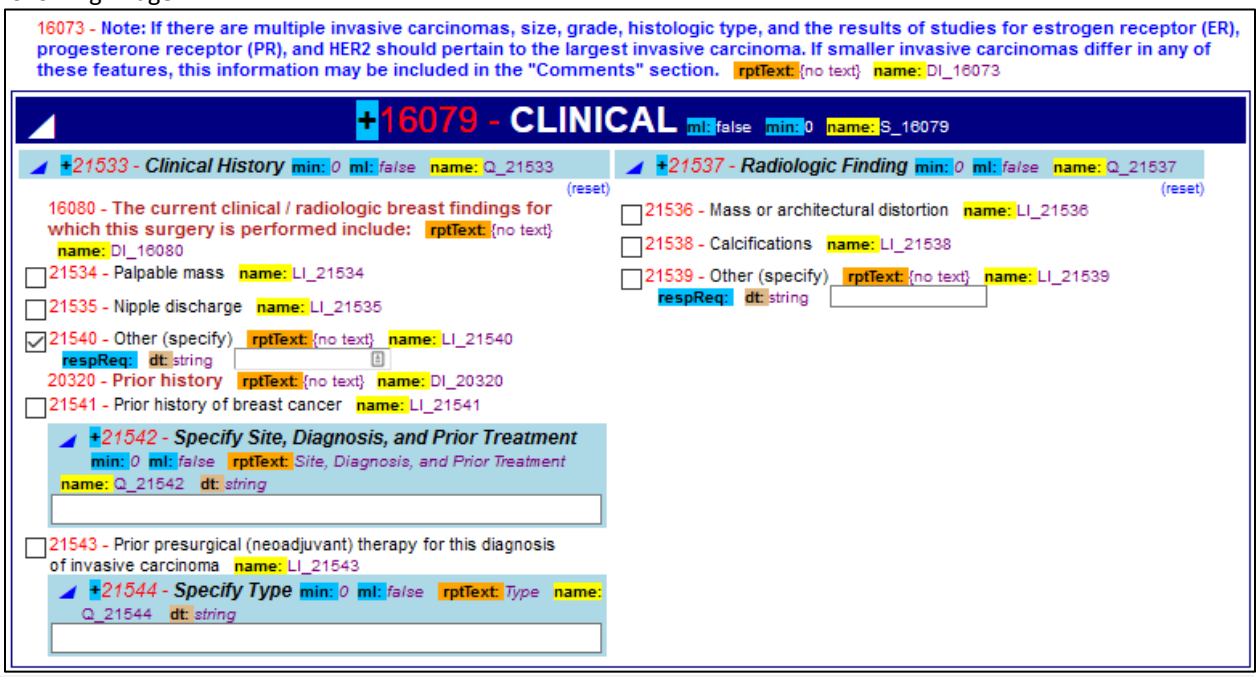


Figure 8: Metadata Icons Displayed in a DEF

The icons from Figure 8 are defined in the table below. The table also shows the item types where the various elements and attributes may appear.

Metadata	Symbol	Default	Description	Applies To[69]
name	name	""	content of name on all elements	all elements
altText Property	altText		value of altText Property, if present	Q
reportText Property	rptText		value of reportText Property, if present	S, Q, LI, DI
Response *datatype*	dt		the datatype element tag under Response, such as string, integer, decimal	QR, LIR
ResponseUnits	un	""	value of ResponseUnits, if populated	QR, LIR
TextAfterResponse	txtAft	""	value of TextAfterResponse, if populated	QR, LIR
Optional to Implement and answer	+		indicates mustImplement = "False" and minCard = "0"	Q, S
mustImplement on ListItem	+	"True"	indicates mustImplement = "False"	LI
mustImplement	ml: false	"True"	indicates mustImplement = "False"	S, Q, LI, BA
minCard	min:	1	content of minCard	S, Q
maxCard	max:	1	content of maxCard, if maxCard <> 1	S, Q
omitWhenSelected	ows	False	indicates omitWhenSelected = "True" (Currently, the ows ListItem's title is also prefixed with "?". This is likely to be removed in a future release.)	LI
responseRequired	respReq	False	indicates responseRequired= "True"	LIR
selected	sel	False	indicates selected = "True"	LI
selectionDisablesChildren	sdac	False	indicates selectionDisablesChildren = "True"	LI
selectionDeselectsSiblings	sds	False	indicates selectionDeselectsSiblings = "True"	QM
readOnly	locked	False	indicates readOnly = "True"	Q

Table 1: Table of Commonly-Used Elements and Attributes.

[69] In this column, LI includes LIR, and Q includes QM, QS and QR.

17 Closing Comments

This document covered substantial parts of the IHE SDC specification. It focused on FDF metadata and its implications for DEF representation and behavior.

However, many aspects of SDC were not covered in depth, including FF, FM, FR and CDE management, FDF modeling standards, the FDF quality review process, DEF implementation techniques, SDC rules, SDC version comparisons, SDC modeling tools, the SDC reference implementation, SDC data transmission and transactions with SOAP and REST API models, the detailed SDC Package model, alternative data storage approaches, reporting, mapping, and querying of SDC-based data. Existing SDC features continue to be refined, and more features (e.g., form behavior rules) are being added. Despite continued SDC development, the SDC concepts covered in this document are very stable and have been tested in several IHE Connectathon with a variety of participating teams.

SDC documentation will improve over time. For now, more information, Schemas, sample FDFs and sample applications can be found in the October 2016 IHE SDC Profile and at the SDC GitHub site. Detailed descriptions of SDC elements and attributes are contained within the SDC Schema files, in the annotation/documentation tags. For assistance with SDC implementations or related questions, please contact the Structured Data Team at the College of American Pathologists: call (847) 832-7700 or email capecc@cap.org.