DIGGS
DATA INTERCHANGE FOR GEOTECHNICAL AND GEOENVIRONMENTAL SPECIALISTS

Developing with DIGGS: A Starter Guide

Contents

DIGGS Overview

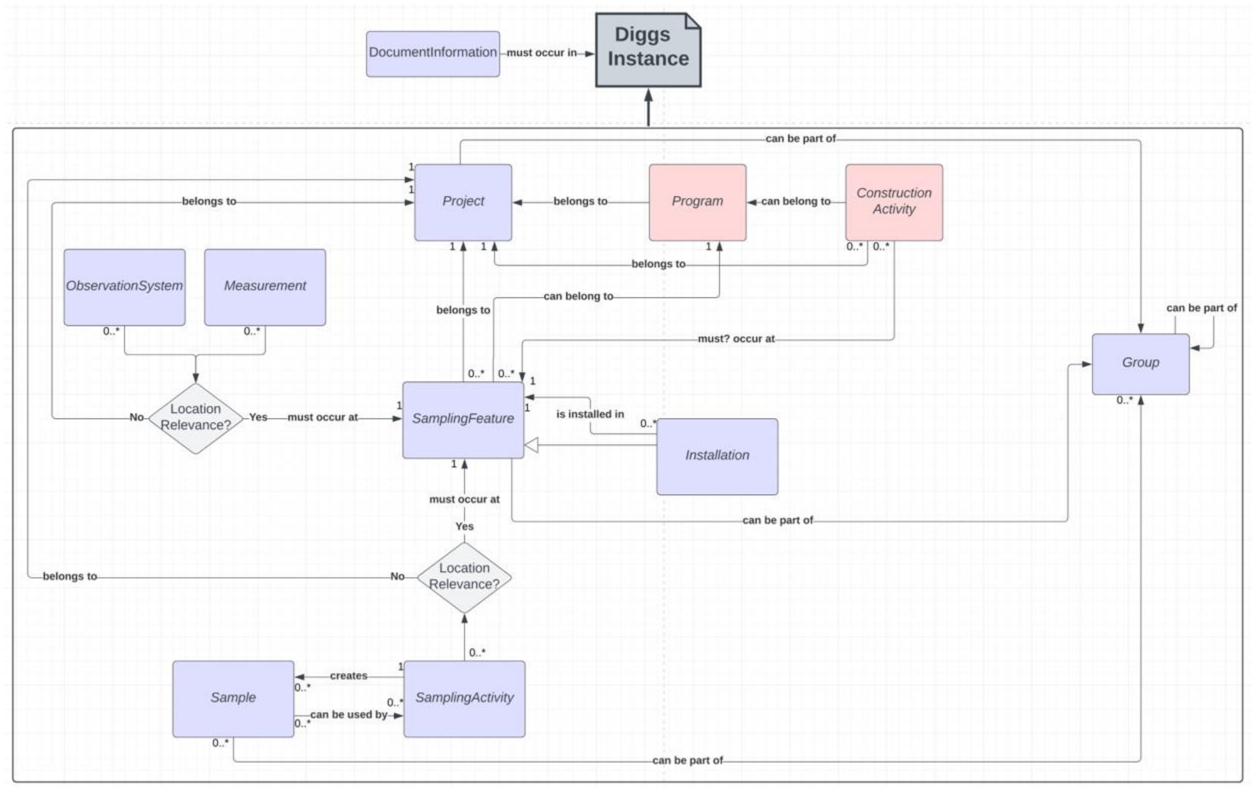
What is DIGGS?

Data Interchange for Geotechnical and Geoenvironmental Specialists (or DIGGS) is a data transfer protocol that uses a relational structure to allow for transferring data between systems or organizations regardless of how that data is ultimately stored or presented. This XML-based standard is designed with both flexibility and robustness in mind allowing it maximum compatibility between disparate geotechnical and geoenvironmental systems for retrieving, storing, or displaying data.

Why should you care?

Prior to the existence of DIGGS, the only way to integrate disparate geotechnical systems was a patchwork system of non-standard implementations of CSVs or JSON. While these prior systems were and are useful for one-off implementations, the lack of a robust modern industry-wide standard necessitates time consuming duplication of integrations as systems change, are swapped out, or upgraded. DIGGS solves this problem. Rather than using patchwork integration systems, DIGGS allows organizations and the systems they use to implement a wheel and spoke integration model, where each individual system produces and consumes the same format of data.

Major DIGGS Objects and Their Relationships



The DIGGS XML schema is composed of different types of object classes that are associated with each other in various ways. These relationships are designed to allow for the various ways in which data are connected. The following section will briefly describe the nature and purpose of the major DIGGS object classes along with how they are connected. More specific information about each of these major objects and how to use them are provided in other sections of this guide.

Project

Projects represent the main business activity under which all data are collected or activities are performed. A project typically is associated with an investigation or some sort of construction activity. All other DIGGS object classes must be associated with a project. In addition to being the object that links all of the information developed within it, the Project object also can hole information on the location of a business activity, the roles of people or entities involved in the project, contract information, or other relevant notes.

Sampling Feature

Sampling features are physical objects or locations through which we observe or measure properties of an investigation target, or perform some sort of activity. It serves to define the dimensionality, extent, and local spatial reference system for zones where observations or activities take place. All sampling features must belong to a project. DIGGS provides several specialized sampling feature objects that are classified by the dimensionality of their geometries. A Station, for example, is a zero-dimensional sampling feature that is represented as a point in space. Borehole, Sounding, Transect, and TrialPit sampling features are one-dimensional features that are represented by a linestring in space. Recent versions of DIGGS have also defined 2D and 3D specialized sampling features to accommodate such sampling spaces as trench walls, cross-sections, and seismic volumes. We will only be discussing the Borehole and Sounding sampling features in this documentation. Installation

Installations are a type of sampling feature, but they differ in that they are contained within another sampling feature. Common examples for this type of object would be having a well installed in a borehole, or an instrument array aligned along a transect. As is nominally implied. installations reference the sampling feature in which they are installed. DIGGS currently defines only one installation object – a Well, which serves to represent piezometers or more complex well constructions.

Sampling Activity

A sampling activity is defined as an action taken to obtain or produce one or more material samples, even if the activity fails to produce a sample (eg. a core run that produces no recovery). Samples produced from a sampling activity may result from collection at a sampling feature, sub-sampling a previously collected sample, aggregation of two or more samples, or a sample created from component materials (such as a grout sample). Sample

DIGGS provides a Sample object, which is used to describe a physical sample of earth material, fluid or gas. which is obtained via a sampling activity for the purpose of observation or testing. Samples may be collected from sampling features, or produced via aggregation, subsampling, or creating a sample in a lab. Each Sample object must be associated with a sampling activity. In addition to the Sample object discussed in this document, the most recent version of DIGGS has added a specialized GroutSample object to carry unique properties associated with grout material.

Measurement

Results which are derived from testing equipment or sensors are recorded as measurements. Results are usually measured or derived numeric values, or they are category assignments which can be calculated from numeric results. DIGGS defines three measurement objects. A Test object, which is used to record location indexed data, such as from laboratory tests such as particle size or Atterberg limits tests on collected samples, or In-situ tests like CPT or SPT. A second type of measurement is a Monitor, which is used to record time-indexed measurements, such as those from a piezometer or inclinometer. The third type of measurement is a MaterialTest, where the observed properties of the measurement refer to a material sample for which there is no location relevance (such as a density measurement done on a grout sample created in the lab).

Measurements, are associated with a sampling feature whenever they are relevant to a particular location, otherwise they are associated with a project.

Observation

Results for observed properties of earth materials that are derived from the use of human senses, judgement, interpretation, or analysis, are recorded in DIGGS as observations. These are generally subjective, usually non-numeric, and typically involve category-based classifications. Some typical examples include color, soil classification, field-based physical property determinations; or other interpretiations that may include the assignment of soil genesis include genesis (eg. alluvium, beach sand), formation assignment (time or spatial ordering), or modeled data (eg. velocity layering, other geotechnical property).

Specific types of observations are organized within DIGGS into several kinds of observation systems as follows:

ColorSystem – used to describe the color(s) of earth materials within a sample or sampling feature as a discrete property

ConstituentSystem – used for observations of constituent components of earth material within a sample or sampling feature (such as the presence of a particular mineral, fossils, cementation, etc.)

DiscontinuitySystem – used to describe the nature and character of faults, fractures and joints and their spacing within a sample or sampling feature

GeoUnitSystem – used to describe observations of a geologic or geotechnical unit where the unit is primarily defined by the value(s) or a value range of one or more physical properties

LithologySystem - used to describe the properties of the primary and any component lithologies, (eg. Silty sand, basalt, etc.)

OrientationSystem – used to define the geometry of vectors or planar surfaces encountered at a sampling feature or in a sample, such as bedding, joints, cross-beds, etc. These observations are designed to characterize regions with generalized geometries, whereas a Discontinuity system would be used to describe discrete features.

OtherObservationSystem - a catch-all used to describe features not specifically accounted for in DIGGS

StratigraphySystem – used to describe ordered bodies of rock or soil, such as formations, biostratigraphic units or aquifers, within a sampling feature or sample.

Only the LithologySystem and GeoUnitSystem (and their component objects) are discussed in this document. Information on the other observation systems can be obtained from the full schema documentation.

Group

Groups are logical associations of projects, sampling features, samples, or other groups where the data from the individual objects may be considered together. An example of this may be a sampling feature group that associates a borehole with an adjacent CPT sounding. Grouping these two sampling features together provides a means of logically associating measurements from the CPT sounding to those from the borehole (such as the lithologic descriptions compared to CPT measurements at equivalent depth in the two features).

Upcoming Additions

DIGGS is an evolving standard with future planned improvements which are actively under development. This guide won't go into detail on how to use these upcoming features as they currently exist only in a preview state, and aspects of how they are designed are subject to change. You are welcome to browse the schema if you're interested in this preview, but it's not recommended to use the below features yet for any type of commercial development.

Construction Activity

An activity intended to alter or improve ground conditions typically executed in accordance with a design specification. Construction activities can either be associated with a program or directly with a project.

Program

Programs contain the properties for the design specification and the post-construction performance results for a collection of construction activities and their related sampling features. Programs are contained within a project.

Related Technologies

XML

- A highly extensible mark-up language designed to be readily readable by both humans and machines. Having a working understanding of XML is necessary for developers working with DIGGS, but this information is out of scope for this guide. W3Schools has an excellent introductory guide that is available for free here.

GML

- Geography Markup Language, or GML is a standardized XML vocabulary for defining various geometries and geographical features. DIGGS is idesigned to be an application schema of GML, as it iextends GML objects and conforms to GML's lexical rules.

- Having an advanced knowledge of GML might provide deeper insights into DIGGS design decisions, but is not necessary as a pre-requisite to using this guide to work with DIGGS. GML language features present in DIGGS are explained as necessary when they come up.

XSD

- DIGGS exists primarily as a set of XSD documents. XSD is a way of defining a particular XML schema that a particular XML document (or XML instance) must adhere to. In order to work with DIGGS, familiarity with XSD isn't strictly necessary, but it can be helpful. More information about XSD can be found here.

XSLT

- Extensible Stylesheet Transformation Language is in the family of related XML technologies. It isn't needed to work with DIGGS, but developers ingesting DIGGS into their system may find its ability to rapidly transform data within an XML document useful. XSLT can be used to turn XML into HTML, CSV, JSON, or many other ways of representing data. This guide won't cover XSLT, but an introduction to the topic can be found here.

Resources

This guide is intended merely as an introduction to DIGGS that covers the most common usage cases or scenarios. Readers who work through this guide in its entirety will have a solid foundation and good working understanding for creating or reading DIGGS XML documents but may want to seek out more niche information specific to their use case. The full documentation for using DIGGS can be found here.

If you are already familiar with the DIGGS XML standard, then the documentation may better serve your immediate needs. That being said, many new users may understandably find the great depth and breadth of the documentation daunting. For this reason, users may want to reference the guide on understanding and using the full DIGGS documentation.

Initializing a DIGGS Document and Creating a DIGGS Project

Formatting a DIGGS Document

Prior to getting into the project itself, let's talk about the outermost structure of the DIGGS document's XML. The data for this guide will be derived from an example borehole log based on the Ohio Department of Transportation (ODOT) which can be found here. Because this is a standard XML document, we'll start with an XML declaration as below:

```xml
<?xml version="1.0" encoding="utf-8"?>
```

This establishes some basic information about the document including the version of the XML and the text encoding we're using. It's helpful to include as it disambiguates how to interpret the document's text. More relevant to DIGGS specifically though, is the "Diggs" root element. All of our DIGGS XML will be placed within this element, either as a direct child node or as a descendant of the main node. It contains a number of important XML namespace declarations that will be used throughout the document.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Diggs xmlns="http://diggsml.org/schemas/2.6"
xmlns:diggs="http://diggsml.org/schemas/2.6"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:g3.3="http://www.open-
gis.net/gml/3.3/ce"
    xmlns:glr="http://www.opengis.net/gml/3.3/lr" xmlns:glrov="http://www.open-
gis.net/gml/3.3/lrov"
    xmlns:diggs_geo="http://diggsml.org/schemas/2.6/geotechnical"
    xmlns:witsml="http://www.witsml.org/schemas/131"
    xsi:schemaLocation="http://diggsml.org/schemas/2.6 ../Diggs.xsd"
gml:id="DocExample">
<!-- Abridged XML -->
</Diggs>
```

While working within a particular version of DIGGS, much of this information can be copied from document to document without having to worry about it, but let's highlight a couple of these. The "gml" prefix is used to import OpenGIS's Geography Markup Language. Attributes and elements with this prefix are defined in that standard. Pertinent portions of GML used in DIGGS will be explained as they come up in this guide. Also worth noting is that the DIGGS schema itself is assigned to the default namespace which is nice as this makes up the bulk of DIGGS XML.

Using Document Information to Define Authorship and Track Changes

Objects and Properties in DIGGS

So, we've set up our DIGGS document with our namespace declarations, we have our "Diggs" root node set up, and now we're almost ready to begin. Below is some barebones XML. For brevity, we've abridged the namespace declarations as they take up a lot of space and aren't important for this example.

```
<Diggs>
    <documentInformation>
            <!-- Abridged XML -->
    </documentInformation>

    <project>
        <Project gml:id="Project_600819">
            <gml:name>WAS-676-2.50</gml:name>
            <!-- Abridged XML -->
        </Project>
    </project>
</Diggs>
```

At first glance, the above may seem strange. We have a "project" element with a single child, "Project." Why not just have "Project" at the top level? Isn't it redundant? To us humans, that may seem logical, but it's actually helpful for programs parsing the document. It's organized this way because DIGGS has something called the"Object-Property Rule."

Elements in DIGGS fall in two categories. First, there are objects with "complex types" which are written in PascalCase (also known as upper camel case). In the above example, the "Diggs" element and the "Project" elements are both complex type objects. Complex types are special because they are allowed to have multiple children, but they must all be properties. This brings us to our second category.

The second category consists of properties of "simple type" or "property type." Properties are written in camelCase (also known as lower camel case). Properties can only contain one child which can be either a string or a complex type object. So, in the above code "gml:name" is a simple type property because it has a string as its only child. Similarly, the "project" element is a "property type" property because it contains only one child "Project" which is a complex type object.

Regardless of which of these categories an element fits into, it does not have a restriction on the number of attributes it can have. You can add any number of valid attributes within the schema.

With this aside out of the way, we're ready to get back into setting up our DIGGS document.

Creating the DocumentInformation Object

Often with any collection of data, it's useful to have information about its origin, and it is in that vein that we'll be using the "DocumentInformation" object. To begin, in our example we want to record when a DIGGS document was generated and who it was generated by. Let's look at some example XML which accomplishes this and walk through it.

```xml
<Diggs>
    <documentInformation>
        <DocumentInformation gml:id="DGS1EB9-15B4-426B-A279-5C653">
            <creationDate>2023-07-26</creationDate>
            <author>
                <BusinessAssociate gml:id="BA_1">
                    <gml:name>Dataforensics</gml:name>
                </BusinessAssociate>
            </author>
            <disclaimer>Not specified in OGC file</disclaimer>
        </DocumentInformation>
    </documentInformation>
</Diggs>
```

Firstly, you may notice we've abridged the namespace declarations to save space. That aside, the "DocumentInformation" complex type object can't be directly within the Diggs complex type object because properties of objects must be of either simple type or property type. So, we have the "documentInformation" property type property which contains our "DocumentInformation" object. If this is still unclear, it might be worth reviewing the previous section.

Shifting gears, our initial goal is to record when the document was created, and we accomplish this through the "creationDate" property which is a simple type property containing only a string. We record the date in the ISO 8601 date format, or YYYY-MM-DD.

Next, in order to record the authorship of the document, we use the appropriately named "author" property type property. It has one child, a "BusinessAssociate" object. For our example, the "BusinessAssociate" just has one child which is the "gml:name" simple type property with the value, "Dataforensics." In our hypothetical scenario, this is the name of the company who developed a tool that generated this DIGGS XML document automatically using some input data. Neat!

Lastly, we've added a disclaimer property to the "DocumentInformation" object which we use to specify that the original source of the raw data, OGC, didn't have authorship information, so some was automatically generated.

While this is all well and good, it'd probably be better to explicitly state that we don't have an original human author so no one gets confused if they look back at this data. Let's use the audit trail to leave a note for anyone checking this in the future. Some updated code with this is below:

```xml
<documentInformation>
    <DocumentInformation gml:id="DGS1EB9-15B4-426B-A279-5C653">
        <creationDate>2023-07-26</creationDate>
        <author>
            <BusinessAssociate gml:id="BA_1">
                <gml:name>Data Forensics</gml:name>
            </BusinessAssociate>
        </author>
        <disclaimer>Not specified in OGC file</disclaimer>
        <auditTrail>
            <Remark>
                <content>Generated by Dataforensics Feedback Tool</content>
                <remarkDateTime>2023-07-26</remarkDateTime>
            </Remark>
        </auditTrail>
    </DocumentInformation>
</documentInformation>
```

Alright! Now we have an “auditTrail” property. This can be especially useful if a DIGGS XML document is iterated on over time by multiple people, as it allows for dated notes. The content for our “auditTrail” entry lives in a Remark object which contains two simple type properties: “content” and “remarkDateTime.” The “content” is used for storing the actual message itself, and “remarkDateTime” is used to record when the note was appended to our XML document. It's best practice to always include both. Even though we have only one 'auditTrail' property here, including both disambiguates the timeline should more entries be added in the future

With this, we have our DIGGS document formatted and ready to go for our project!

What is a Project?

Projects are the overall wrapper for the DIGGS document which encompasses the main business activity. This usually is associated with an investigation or some sort of construction activity. All other DIGGS object classes must belong to a project. The project primarily exists for use as this umbrella, but it also can be used to house data such as the location of a business activity, the roles of people or entities involved in the project, or other relevant notes as we’ll demonstrate below.

Creating a Project in DIGGS

Alright, so in this hypothetical example, we’ll be creating a DIGGS project to record the data from a project on Route 676, an existing road to which some improvements are planned. Let’s begin with some barebones example XML:

```xml
<Diggs>
    <project>
        <Project gml:id="Project_600819">
                <gml:description>Route 676 Improvements</gml:description>
                <gml:name>WAS-676-2.50</gml:name>
                <gml:name codeSpace="ODOT Structure File Number">N/A</gml:name>
                <gml:name codeSpace="ODOT PID">111644</gml:name>
                <internalIdentifier codeSpace="OGC">600819</internalIdentifier>
        </Project>
    </project>
</Diggs>
```

Okay, so the “Project” object is the workhorse here, but it lives within the project property of our root Diggs object. As before, we’ve omitted the namespace declarations to save space, but they are still required.

We’re using the appropriately named gml:description to add a human readable description of what the project is about. Ideally, we want to be able to look at this at a glance and reveal which project this document pertains to. We also have a number of pieces of identifying information recorded in the gml:id attribute, the gml:name properties, and the internalIdentifier property. These identifiers are recurring properties you’ll find throughout the DIGGS document, and more information on them can be found here.

With our identification information set up, we’ll pivot to answering the questions, when and where did this project take place? Let’s take a look at code above with some new additions:

```xml
<project>
    <Project gml:id="Project_600819">
        <gml:description>Route 676 Improvements</gml:description>
        <gml:name codeSpace="ODOT PID">111644</gml:name>
        <internalIdentifier codeSpace="Ohio State ID">600819</internalIdentifier>

        <projectDateTimeSpan>
            <TimeInterval gml:id="TI-1">
                <start>2021-01-11</start>
                <end>2021-01-20</end>
            </TimeInterval>
        </projectDateTimeSpan>

        <locality>
            <Locality gml:id="DGS4FEA-70-4042-6759-F55C">
                <address>
                    <Address gml:id="Project_600819_address">
                        <county>WAS</county>
                    </Address>
                </address>
                <route>676</route>
                <segment>2.50</segment>
            </Locality>
        </locality>
    </Project>
</project>
```

Starting with the question of when, we're using a "projectDateTimeSpan" property to track the start and end dates. We do this with a single child, a "TimeInterval" object with two properties: start and end. Each uses the same ISO 8601 date format (YYYY-MM-DD) that we're using throughout the document.

Secondly, let's address where this occurs. Our "Project" object has a "locality" property with a single child, a "Locality" object. These similarly named tags are consistent with the "Object-Property" rule described previously in this document. In this case, we'll just be recording a few pieces of information. Our source borehole log identifies the location of the project with the county, the route (highway), and which section of that highway the work takes place on, so we'll be using all that information.

The route goes in the intuitively named "route" property, but the section of the road is recorded on a property new to DIGGS 2.6 "segment." Both are simple type properties with a string as their value. Recording the county is something we do within an "Address" complex type object stored within the "address" property. In this case, the "Address" object has only one child, a "county" property, but there are a number of other possible properties that can be used in an address to suit your needs. These include "number," "streetAddress," "city," "state," "province," "country," and "postalCode." Information about properties such as these can be found in the full documentation here. A breakdown of how to use the full DIGGS documentation can be found here.

So now we've answered the question of where and when, but we haven't addressed who. Let's give out some credit. We'll be doing this with roles. These will be used elsewhere in DIGGS, but let's look at how we can use them to describe people and entities associated with a project:

```
<project>
    <Project gml:id="Project_600819">
        <!-- Abridged XML -->
        <role>
            <Role>
                <rolePerformed>Client</rolePerformed>
                <businessAssociate>
                    <BusinessAssociate gml:id="DGS616C-16BD-41B8-E1F7-50EB8">
                        <gml:name>ODOT</gml:name>
                    </BusinessAssociate>
                </businessAssociate>
            </Role>
        </role>
        <projectDateTimeSpan>
            <!-- Abridged XML -->
        </projectDateTimeSpan>
        <locality>
            <!-- Abridged XML -->
        </locality>
    </Project>
</project>
```

Projects can have any number of "role" properties, between zero and infinity. Because each "role" is a property, it can only have one child which will invariably be a "Role" object. The "Role" object has a number of possible properties, here we're using two. First, "rolePerformed" is a simple type property which describes the work that was performed or the relationship of the entity to the project. In this case, we're recording that the client for this project is the Ohio Department of Transportation (ODOT). To specify ODOT, we're using the "businessAssociate" property which has an object as a child, "BusinessAssociate." Within this object, we utilize the 'gml:name' property to designate ODOT as our client. While this is sufficient for our example, additional properties such as "title," "address," "emailAddress," or "phoneNumber" could also be included to be more comprehensive.

Let's look at a couple more examples:

```xml
<project>
    <Project gml:id="Project_600819">
        <!-- Abridged XML -->
        <role>
            <Role>
                <rolePerformed>Project Engineer</rolePerformed>
                <businessAssociate>
                    <BusinessAssociate gml:id="DGS86C0-1625-33AF-16B3-23D1E">
                        <gml:name>Dataforensics</gml:name>
                    </BusinessAssociate>
                </businessAssociate>
            </Role>
        </role>
        <role>
            <Role>
                <rolePerformed>Contractor</rolePerformed>
                <businessAssociate>
                    <BusinessAssociate gml:id="DGS455-241-1385-AE1D-55FB9">
                        <gml:name>Big Dog Drillers</gml:name>
                    </BusinessAssociate>
                </businessAssociate>
            </Role>
        </role>
        <!-- Abridged XML -->
    </Project>
</project>
```

These roles follow the same structure we've previously established but are included to demonstrate how roles can be employed to identify various stakeholders in a project. With this setup, our DIGGS document is now ready for the metaphorical heavy lifting! If you've been following this guide, you can compare your work to an unabridged version of the sample code, which is available here. Assuming you're progressing through this guide in sequence, the next topic will be Logging a Borehole and CPT Sounding.

Using DIGGS Identifiers

Introduction

While using DIGGS, including as you follow along with the guide here, you'll notice some especially important recurring elements that are inherited from Geography Markup Language (GML). Rather than repeat this information in each article in which it is needed, it can be found below.

Gml:id

The gml:id attribute is used for assigning a unique identifying value for each object in a DIGGS XML document. The point of emphasis here is its uniqueness. No two IDs within a document may share a value, and although not enforced by the DIGGS XML standard, you should endeavor not to re-use the same IDs across different XML documents. Maintaining persistence of your gml:id values helps to ensure clean and traceable data. The reason this unique quality is so important, is that it is used within DIGGS as a key value to associate different objects, including objects which don't have a hierarchical relationship with each other. More information about how these associations can be made is provided elsewhere in this guide, but for now it's enough to understand that this is the value necessary for making such a connection.

There are various ways to ensure that each gml:id is both persistent and unique. Below are some tips for methods to maintain persistence:

- The authority providing the data stores the gml:id within an internal storage system, such as a database, which is referenced prior to transmitting data to or from DIGGS.
- A formula is used to assign an ID in such a way that key information such as an internal identifier can be extracted by parsing the ID when importing data from DIGGS.
- Each data provider instead uses a gml:identifier (explained below) for tracking persistence within their respective system.

Additionally, below are some tips for ensuring uniqueness when assigning gml:id values so that they aren't repeated across documents:

- Data providers can store assigned gml:id values with an association to an internal key. Whenever a new gml:id has to be assigned, a check can be performed against previously used IDs to ensure none are repeated.
- Generators can be used to assign UUIDs (or GUIDs) as a portion of the ID.
- Additionally, it's suggested that a code corresponding to the authority creating the DIGGS document is added to the ID to easily determine the source of uniqueness and prevent the likelihood of ID collisions between different organizations or systems within an organization.

Using a combination of the above tips will help ensure that you have valid DIGGS XML with unique persistent IDs that make associating and querying data useful and easy.

gml:Name

The gml:name attribute provides a way to label or identify an object, commonly using a human readable descriptive name. An object may or may not have multiple names in which case these names are assigned to different “authorities” using the gml:codeType attribute. For example, a field technician (LOGR) may assign a name to a sample which then gets another name assigned once it’s been processed by a lab (ILA). The abridged code can be found below:

```
<Sample gml:id="LOGR-uniquesampleid">

   <gml:name codeSpace="LOGR">BH-1-35</gml:name>

   <gml:name codeSpace="ILA">GL-1443</gml:name>

</Sample>
```

In the above example, the “LOGR” codeSpace value is associated with a hypothetical tablet logging application used at the job site, and the “ILA” codeSpace value is associated with a hypothetical LIMS system used to log results from an assay being run on the sample. By maintaining both names under different codespaces, each system can be tuned to display the name relevant to that system without losing the association to the other system. Finally, it should be noted that each “authority” may or may not require names to be unique, so the DIGGS standard itself does not enforce this requirement. It is instead up to the implementation to ensure that any unique naming constraints are respected.

Gml:Identifier

The gml:identifier element is similar in many respects to the gml:id attribute. Both are used to assign values for the purpose of identifying an associated object. The difference is that while gml:id values are used to link DIGGS objects and have restricted possible values, the gml:identifier’s possible values are more flexible. Being able to, as an example, start with a letter, means it can store a UUID (or GUID) without needing to append a The prefix. Similar to the gml:name attribute, gml:identifier uses the codeSpace attribute to identify the controlling authority for the identifier. An object in DIGGS may have multiple names, but it may only have one gml:identifier. This is because the gml:identifier’s intended use is to communicate the key value belonging to the entity originally transmitting the data.

Boreholes and CPT Soundings

Introduction

In this coming section, we are going to continue working with an example borehole log based on the Ohio Department of Transportation (ODOT) which can be found here. Prior to working through this document, you should already know how to set up a DIGGS document with the necessary namespace declarations, along with the “DocumentInstance” and “Project” objects. The examples found later in this guide will reference back, as appropriate, to XML written in the previous section of this guide. If this isn’t already familiar to you, more information on setting up your DIGGS document can be found here.

Working with Sampling Features

What is a Sampling Feature?

Sampling features are physical objects or locations used to measure or observe the properties of an investigation target or to perform some sort of activity. It serves to define the dimensionality, extent, and local spatial reference system for zones where observations or activities take place. All sampling features must belong to a project. Examples of sampling features include boreholes or trench walls.

The sampling features we create for our borehole and CPT soundings will not contain all the information about them but instead focus on the aspects such as the location, people involved, and its dimensions. More in depth information appears elsewhere and will be covered as it comes up.

Boreholes as Sampling Features

To save space, we have excluded the namespace declarations, the document instance, and the project from these examples in order to save space, but they can be seen in full here. Our “Diggs” root element can have multiple “samplingFeature” properties, and in most cases will. Like with all properties, each “samplingFeature” can only have one child which in this case is going to be a “Borehole” object. Let’s look at some starting code:

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <gml:name>B-001-0-20</gml:name>
        <internalIdentifier codeSpace="OGC">B-001-0-20</internalIdentifier>
        <status>Final</status>
      </Borehole>
</samplingFeature>
```

First and foremost, we need to identify our borehole. We have grabbed, “B-001-0-20” from our log and automatically appended, “Location_” as a prefix for a couple reasons. First, it guarantees our gml:id will start with a letter as is required, but secondly it also has the advantage of our being able to tell at a glance what this ID is for. The value “B-001-0-20” is guaranteed to be unique within the project thanks to tracking of internal persistence from the data source. Since this DIGGS document is only going to contain one project, we can count on this “gml:id” not colliding with another ID. We are going to be referencing this “gml:id” attribute elsewhere in our code, so it’s an especially important one.

There are also a few other important properties we arere starting with. All “Borehole” objects must have a name, so here we’ve set the name to the same value as the ID. Since our source log lacks a human-readable name for the borehole, this ID will suffice. It similarly appears as the value for our “internalIdentifier” property. In this case, we are not appending the “Location_” prefix to the value because we aren’t concerned about restrictions on the string, and we want to maintain parity between this value and the original source of the data. We have appended the “OGC” “codeSpace” attribute to the “internalIdentifier” to denote that its origin is in OpenGround. As for our, “status” property, this is a string that tracks where in the process this borehole is in terms of operations, review, or quality concerns. Our example is an historical one that is completely finished, so it’s marked as “Final.”

Next up, we’re going to add roles for the logger, driller, and the drilling contractor as below:

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <gml:name>B-001-0-20</gml:name>
        <internalIdentifier codeSpace="OGC">B-001-0-20</internalIdentifier>
        <status>Final</status>

        <role>
            <Role>
                <rolePerformed>Logger</rolePerformed>
                <businessAssociate>
                    <BusinessAssociate gml:id="DGS4DF9-330-41FA-9E28-4B844">
                        <gml:name>Williams</gml:name>
                    </BusinessAssociate>
                </businessAssociate>
            </Role>
        </role>
        <role>
            <Role>
                <rolePerformed codeSpace="Diggs">Drilling Contractor</rolePer-
formed>
                <businessAssociate>
                    <BusinessAssociate gml:id="DGS418F-493-1C3C-665E-65EC6">
                        <gml:name>ODOT</gml:name>
                    </BusinessAssociate>
                </businessAssociate>
            </Role>
        </role>
        <role>
            <Role>
                <rolePerformed codeSpace="Diggs">Driller</rolePerformed>
                <businessAssociate>
                    <BusinessAssociate gml:id="DGS8550-1150-D7F-B5B5-48AB7">
                        <gml:name>Carey</gml:name>
                    </BusinessAssociate>
                </businessAssociate>
            </Role>
        </role>
    </Borehole>
</samplingFeature>
```

You'll notice these role properties follow the same format as they did in the Project. They are the same DIGGS object, they are just being used to refer to who is responsible for a particular borehole instead of aspects of the overall project. As these roles work the same way as they do in the project, we won't cover them in depth. As this example is getting a bit long, the previous XML will be abridged in further examples for the sake of clarity and readability.

Next up, are the “investigationTarget” and the very important project reference, “projectRef.” Both of these are required, so let’s look at some XML:

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
    </Borehole>
</samplingFeature>
```

The “investigationTarget” property uses a string to indicate the target of the investigation, but the list of possible values is restricted to one of the following:

- Deep Foundation - investigation of deep foundation elements
- Earthworks -earth materials excavated and/or emplaced by engineering activity
- Ground Water -hydrology and chemistry of ground water;
- Indoor Atmosphere - meteorology and chemistry of air in indoor spaces
- Natural Ground - earth materials sampled in natural or undisturbed state;
- Outdoor Atmosphere - meteorology and chemistry of air in outdoor spaces
- Surface Water - hydrology and chemistry of surface water;

Not all of these possible values are appropriate for a borehole obviously but may apply for “investigationTargets” of other features. In our case, we’re examining undisturbed ground adjacent to a highway, so the value “Natural Ground” is chosen.

As for the, “projectRef” property, this is very important for our “Borehole” in DIGGS. All sampling features must be linked to a project, and this is how we establish that connection. We use an “xlink:href” attribute with the value set to the “gml:id” of the project appended with a, “#” character.

Up next, we will be recording where this borehole is. Let’s work through some more example code:

```
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
        <locality>
            <Locality gml:id="DGS5FC9-1518-2B95-6022-3CD8B">
                <route>CL SR 676</route>
                <station>131+46</station>
                <offset uom="ft">7</offset>
                <offsetDirection>RT</offsetDirection>
            </Locality>
        </locality>

        <referencePoint>
            <PointLocation gml:id="pl-B-001-0-20">
                <gml:pos srsDimension="3"
srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
                    uomLabels="degrees">39.474660 -81.796858 818.6</gml:pos>
            </PointLocation>
        </referencePoint>
    </Borehole>
</samplingFeature>
```

Firstly, there is now a “locality” property with a “Locality” object as its child. This is the same type of object we saw previously in the “Project” object, but we’re using different properties here to record the borehole’s location. The “route” property is being used to record the highway in question, and the “station” property similarly records the station. The offset is placed in its own property “offset” with a unit of measure attribute “uom” which we mark as “ft” to notate that the value is in feet. The possible values we can use as a unit of measure are defined in a list which can be found in the full documentation here. Finally, the “offsetDirection” property is used to record that it is from the right, or “RT” as it’s notated.

Next, the “referencePoint” property’s purpose is to note the position of, in this case, the borehole. We do that with a property from GML: “gml:pos.” A lot of work is being done here with the attributes; “srsDimensions” is set to “3” as we will be using latitude, longitude, and altitude. This format for our special reference system is referenced with our special reference system (srs) name attribute, “srsName.” The above value is denotes we are using WGS284 latitude and longitude for the first two dimensions then NGVD29 height in feet for the third. Finally, the actual latitude, longitude, and elevation are written in order as the value of the “gml:pos” object, separated by spaces.

Following this, we will be describing the geometry of our borehole using the “centerLine” property which is expanded upon with the “linearReferencing” property. As previously, an additional portion of the code will be abridged in the below example to improve clarity:

```
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <referencePoint>
            <PointLocation gml:id="pl-B-001-0-20">
                <gml:pos srsDimension="3"
srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
                    uomLabels="degrees">39.474660 -81.796858 818.6</gml:pos>
            </PointLocation>
        </referencePoint>
        <centerLine>
            <LinearExtent gml:id="cl-B-001-0-20">
                <gml:posList srsDimension="3"
                    srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
                    uomLabels="degrees">39.474660 -81.796858 818.6 39.474660 -
81.796858 777.6
                </gml:posList>
            </LinearExtent>
        </centerLine>
    </Borehole>
</samplingFeature>
```

Now we have a new "centerLine" property with a "LinearExtent" object to define the geometry of our borehole. It's similar to the previous "referencePoint" property, but there are some key differences. Our "LinearExtent" object has a "gml:postList" property which represents a list of positions. We need to use one for the top of the borehole and one for the bottom. It has the same attributes with the same values as our "gml:pos" object from the previous example because we're using the same coordinate system for mapping the borehole. The value of the "gml:posList" is also a space separated list, but the first three values are for the top of the borehole, and the second set of three values are for the bottom.

It is also worth stopping to note that the "gml:id" for the "LinearExtent" object has been defined by using the "internalIdentifer" of the borehole then adding a "cl-" prefix for center line. As there is only one center line for each borehole, this ensures each borehole has a unique ID thanks to borehole ID persistence being managed by the host system. All "gml:id" values need to be unique, but this one is worth pointing out as it will soon be referenced.

Next, we will be looking at the "linearReferencing" object:

```
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <centerLine>
            <LinearExtent gml:id="cl-B-001-0-20">
                <gml:posList srsDimension="3"
                    srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
                    uomLabels="degrees">39.474660 -81.796858 818.6 39.474660 -
81.796858 777.6
                </gml:posList>
            </LinearExtent>
        </centerLine>

        <linearReferencing>
            <LinearSpatialReferenceSystem gml:id="lsr-B-001-0-20">
                <gml:identifier codeSpace="urn:x-def:authority:DIGGSINC">urn:x-
diggs:def:fi:DIGGSINC:B-001-0-20-lsr</gml:identifier>
                <glr:linearElement xlink:href="#cl-B-001-0-20"/>
                <glr:lrm>
                    <glr:LinearReferencingMethod gml:id="lrm-B-001-0-20">
                        <glr:name>chainage</glr:name>
                        <glr:type>absolute</glr:type>
                        <glr:units>ft</glr:units>
                    </glr:LinearReferencingMethod>
                </glr:lrm>
            </LinearSpatialReferenceSystem>
        </linearReferencing>
    </Borehole>
</samplingFeature>
```

In the above code, the "LinearSpatialReferenceSystem" can be seen. It is used within our borehole to contextualize the information of our "LinearExtent" object in our "centerLine." The "gml:identifier" is required for this object, and we use it to denote that DIGGS is defining this information. Next, we use the, "glr:linearElement" property to connect this object back to its "LinearExtent" using the "xlink:href" attribute. As before, whenever using an href we append a "#" prefix to the gml:id of the target object.

We define the linear referencing method inside the "glr:lrm" property with a "glr:LinearReferencingMethod" object. It has three properties, "glr:name," "glr:type," and "glr:units," which we fill with chainage, absolute, and feet ("ft") as is appropriate to our use case.

Now that we've defined the basic geometry of the hole, we will shift to some other pertinent information:

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <whenConstructed>
            <TimeInterval gml:id="DGS86CE-1566-567-125B-5E790">
                <start>2021-01-11T00:00:00</start>
                <end>2021-01-12T00:00:00</end>
            </TimeInterval>
        </whenConstructed>
        <totalMeasuredDepth uom="ft">41.00</totalMeasuredDepth>
        <boreholePurpose>Landslide</boreholePurpose>
        <boreholeType>BH</boreholeType>
    </Borehole>
</samplingFeature>
```

First in this section, when was the borehole created? This is recorded using the “whenConstructed” property with a “TimeInterval” object. The “TimeInterval” has two properties “start” and “end” which we use to record the starting and ending as a datetime string. Next, the total depth is recorded in the, “totalMeasuredDepth” property with a unit of measure “uom” attribute set to feet (“ft”). In this case, our borehole is forty-one feet deep. Following this, we record the purpose for drilling the borehole in the “boreholePurpose” property; in this case we are investigating the quality of the soil in the aftermath of a landslide that occurred nearby. Finally, the “boreholeType” is used to specify general information about the construction of the borehole such as mud rotary or reverse circulation. In our ODOT example, this is generically referenced as just, “BH.” In another example, an organization might have a list of codes for different types of boreholes. This list would be referred to by adding a “codeSpace” attribute with a corresponding reference.

Transitioning into the next section, we will have a series of properties that describe the depths of different techniques used to drill the borehole and events that occurred as a result of that process. We begin this process with the ending, backfilling the borehole:

```xml
<backfill>
    <Backfill gml:id="DGSC58-72-2718-1A74-25687">
        <backfillLayer>
            <BackfillLayer gml:id="DGS7C0B-15F4-332E-1168-5E2D4">
                <backfillInterval>
                    <LinearExtent gml:id="DGS7DAF-2EC-2428-9296-31AC6">
                        <gml:posList
                            srsDimension="1"
                            srsName="#lsr-B-001-0-20">0.00 41.00
                        </gml:posList>
                    </LinearExtent>
                </backfillInterval>
                <backfillMaterial>Bentonite</backfillMaterial>
            </BackfillLayer>
        </backfillLayer>
    </Backfill>
</backfill>
```

This “backfill” property is also within the borehole object, but the “samplingFeature” and “Borehole” elements have been excluded from this XML snippet to preserve horizontal space. In our example ODOT scenario, it was necessary to completely backfill the borehole after conducting the test, and they decided to use bentonite to do so. So, we have our “Backfill” object which in this case has just one property, “backfillLayer.” We do not need multiple layers because the entire hole was filled with bentonite. We define the height of our single layer using a “LinearExtent” object within the “backfillInterval” property. We have used this object before, but this time we only have one dimension, so “srsDimension” is set to one. We use the “srsName” property to reference back to the “linearReferencing” object of this borehole. This is not a proper href though, and the connection is just made for reference. The extent of the backfill is then defined as “0.00 41.00” which is a space separated list denoting the start and end of the backfill. We do not need to define the units as it is already defined in the borehole’s linear spatial reference system. Because our hole is only 41 feet deep, this means the entire borehole has been filled. Finally, we use the “backfillMaterial” property of the “BackfillLayer” to provide the material used which is bentonite.

With this, we have defined how we backfilled our borehole, but we have not yet described what happened as the borehole was being dug. Let’s remedy that:

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <constructionMethod>
            <BoreholeConstructionMethod gml:id="DGS97C7-B38-3F96-8919-53CB5">
                <gml:name>Soil Drill Method</gml:name>
                <location>
                    <LinearExtent gml:id="cm1-B-001-0-20">
                        <gml:posList
                        srsDimension="1"
                        srsName="#lsr-B-001-0-20">0.00 21.00</gml:posList>
                    </LinearExtent>
                </location>
                <constructionMethod>
                    <Specification gml:id="DGS79CD-6E4-1C6B-E15E-B45D">
                        <gml:name>3.25&quot; HSA</gml:name>
                        <shortMethodName>Soil Drill</shortMethodName>
                    </Specification>
                </constructionMethod>
                <constructionEquipment>
                    <Equipment gml:id="DGS48EA-49A-382C-C54-4A13F">
                        <gml:name>CME 55 Truck</gml:name>
                        <class>Drill Rig</class>
                    </Equipment>
                </constructionEquipment>
            </BoreholeConstructionMethod>
        </constructionMethod>
    </Borehole>
</samplingFeature>
```

Here, we can see that the first twenty-one feet of the borehole was dug using a soil drill. Our “Borehole” object has a series of “constructionMethod” properties ordered by the depth they cover. We are just looking at the first such property above, but each one has a “BoreholeConstructionMethod” object as its child. Within the “BoreholeConstructionMethod,” there are three key pieces of information that we are record which follow the “gml:name” property, a property used to record the method name.

The first of these key pieces is within “location” property. It houses a one dimensional “LinearExtent” object as above that defines the span of depth that was dug with this drilling method. It works the same way as been covered previously, so we will not go into great detail here again.

The second key is defining more specifically the method used to drill. It is important to note that it has the same name, “constructionMethod” as its grandparent element. This may seem somewhat self-referential, but what is important to understand for the sake of this example, is that we are using this nested “constructionMethod” property with a “Specification” object to describe the drill used. The “gml:name” property specifies that a 3.25” hollow stem augur (HSA) was used, but because double quote is a special character in XML, it has been encoded as “&quot;” here. Finally, the “shortMethodName” property is used to describe the tool in question which is a soil drill.

As for the third key, we use the “constructionEquipment” property to describe the piece of equipment used to dig the borehole. In our case, a truck with an attached drill rig. We do this with the “Equipment” object using two properties: “gml:name,” which describes the “CME 55 Truck” in question, and the “class” attribute which more generically specifies that this is an oil rig.

The XML for the next and final section of the borehole looks largely the same, but in this case, rock was being drilled through rather than soil. An excerpt is included below:

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <constructionMethod>
            <BoreholeConstructionMethod gml:id="DGS3740-F91-2942-FD4B-41E21">
                <gml:name>Rock Core Method</gml:name>
                <location>
                    <LinearExtent gml:id="cm2-B-001-0-20">
                        <gml:posList
                        srsDimension="1"
                        srsName="#lsr-B-001-0-20">21.00 41.00</gml:posList>
                    </LinearExtent>
                </location>
                <constructionMethod>
                    <Specification gml:id="DGS4AAB-167-26A5-1CC0-2B8E7">
                        <gml:name/>
                        <shortMethodName>Rock Core</shortMethodName>
                    </Specification>
                </constructionMethod>
                <constructionEquipment>
                    <Equipment gml:id="DGS13EA-FC9-4ED7-39C0-43B43">
                        <gml:name>CME 55 Truck</gml:name>
                        <class>Drill Rig</class>
                    </Equipment>
                </constructionEquipment>
            </BoreholeConstructionMethod>
        </constructionMethod>
    </Borehole>
</samplingFeature>
```

You might note that it is largely the same except that it covers feet 21 through 41 and involved drilling through rock.

Next, we will cover how to notate reaching the end of the borehole. It’s recorded as a “constructionEvent” as below:

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <constructionEvent>
            <BoreholeEvent gml:id="DGS9E54-310-4E70-6A6D-32203">
                <location>
                    <PointLocation gml:id="be2-B-001-0-20">
                        <gml:pos
                            srsDimension="1"
                            srsName="#lsr-B-001-0-20">41.0</gml:pos>
                    </PointLocation>
                </location>
                <infoRecorded>
                    <Parameter>
                        <parameterName>End of Boring</parameterName>
                        <parameterValue>EOB</parameterValue>
                    </Parameter>
                </infoRecorded>
            </BoreholeEvent>
        </constructionEvent>
    </Borehole>
</samplingFeature>
```

When entering a borehole into DIGGS, we include a “constructionEvent” property with a “BoreholeEvent” object to record having reached the end of the borehole. This may seem redundant, but it makes referencing this important piece of information easier in an automated fashion. First, we have a “location” property with a “PointLocation” object. It contains a “gml:pos” attribute that is one dimensional and records only one piece of data, the depth of the hole. The unit of measure is implied as before by the borehole’s spatial reference system.

Secondly, we need to notate that this event is for the end of boring which we do with the “infoRecorded” property. Its child is a “Parameter” object with a “parameterName” and a “parameterValue” properties. Here we record the literal values “End of Boring” and “EOB” respectively.

In drier circumstances, this might have been the end of our borehole’s sampling feature, but we have one more event to record: hitting water.

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <waterStrike>
            <WaterStrike gml:id="DGS560C-1576-1C7D-4122-28765">
                <initialWaterStrikeReading>
                    <WaterStrikeReading gml:id="DGS9E54-310-4E70-6A6D-32204">
                        <status>At time of drilling</status>
                        <dateTime>2021-01-11T00:00:00</dateTime>
                        <waterLocation>
                            <PointLocation gml:id="DGS9112-185B-ADE-ADD0-2C977">
                                <gml:pos
                                srsDimension="1"
                                srsName="#lsr-B-001-0-20">6.00</gml:pos>
                            </PointLocation>
                        </waterLocation>
                        <bottomCasing/>
                    </WaterStrikeReading>
                </initialWaterStrikeReading>
            </WaterStrike>
        </waterStrike>
    </Borehole>
</samplingFeature>
```

We add this water strike as a “waterStrike” property of the borehole. Its child, “Waterstrike” contains two properties, “initialWaterStrikeReading” and “postWaterStrikeReading,” which are structured in the same way but occur at two different times. The “WaterStrikeReading” object has a few important properties for our example. The “status” records the status when the strike occurred, in this case at the time of drilling. The “datetime” property records the date and time of the incident. In our example, the operator recorded only the date, so the time is set to midnight to indicate it wasn’t specified. Finally, we use a “PointLocation” object for our “waterLocation” property as before to record the depth at which water was struck.

The postWaterStrike is formatted in the same way, but it occurs a day later.

```xml
<samplingFeature>
    <Borehole gml:id="Location_B-001-0-20">
        <!-- Abridged XML -->
        <waterStrike>
            <WaterStrike gml:id="DGS560C-1576-1C7D-4122-28765">
                <initialWaterStrikeReading>
                    <WaterStrikeReading gml:id="DGS9E54-310-4E70-6A6D-32204">
                        <status>At time of drilling</status>
                        <dateTime>2021-01-11T00:00:00</dateTime>
                        <waterLocation>
                            <PointLocation gml:id="DGS9112-185B-ADE-ADD0-2C977">
                                <gml:pos
                                srsDimension="1"
                                srsName="#lsr-B-001-0-20">6.00</gml:pos>
                            </PointLocation>
                        </waterLocation>
                        <bottomCasing/>
                    </WaterStrikeReading>
                </initialWaterStrikeReading>
                <postStrikeReading>
                    <WaterStrikeReading gml:id="DGSA674-85F-2703-F085-2D612">
                        <status>One day following drilling</status>
                        <dateTime>2021-01-12T00:00:00</dateTime>
                        <waterLocation>
                            <PointLocation gml:id="DGS35DA-14B4-19E9-8679-3EEE0">
                                <gml:pos
                                srsDimension="1"
                                srsName="#lsr-B-001-0-20">7.00</gml:pos>
                            </PointLocation>
                        </waterLocation>
                        <bottomCasing/>
                    </WaterStrikeReading>
                </postStrikeReading>
            </WaterStrike>
        </waterStrike>
    </Borehole>
</samplingFeature>
```

It looks like a day later the water had receded to a foot deeper. It is worth noting that, as before, we do not have to define the units being used in our “PointLocation” “gml:pos” or, “gml:linearExtent” properties as it is already defined at the borehole level.

With this, we have finished creating the SamplingFeature for our borehole! You may have noticed that this didn't occur all aspects of data related to the borehole. That is still coming! But first, we will look at a CPT Sounding.

CPT Soundings as Sampling Features

If you are reading this section having not read the previous section on boreholes, it is recommended that you first read that section. Recording this portion of data for a sounding is similar to that of a borehole, so the foundational knowledge of learning about boreholes first will help you here.

As before, we're leaving out the root "Diggs" node to preserve horizontal space, but the, "samplingFeature" property below is a property of the root, "Diggs" object. As a starting example, here is a baseline that will be built upon:

```xml
<samplingFeature>
    <Sounding gml:id="df-cpt_C-10">
        <gml:name>C-10</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819"/>
    </Sounding>
</samplingFeature>
```

Instead of a "Borehole" object, we are using a "Sounding" one. Similar to before, we're deriving the ID from our example ODOT borehole log, but in this case, we don't have as useful of a persistent ID managed by a third-party system, so we instead take advantage of the names of each sounding being assigned a unique value to define a "gml:id" based on the sounding name. This in turn is carried in its base form as the value for our, "gml:name" property.

Our source data did not record who conducted this sounding, so no roles are included here, but if they were they would be in the same place here as they were for our borehole. As for the "investigationTarget" and the "projectRef" properties, they are the same as before as this CPT Sounding is a part of the same project and was conducted on the same site nearby on the same type of ground.

Next, we will look at the remaining code for the sounding:

```xml
<samplingFeature>
    <Sounding gml:id="df-cpt_C-10">
        <gml:name>C-10</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
    </Sounding>
    <referencePoint>
        <PointLocation gml:id="pl-C-10">
            <gml:pos srsDimension="3"
            srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
            uomLabels="degrees">39.475026 -81.795909 828.80</gml:pos>
        </PointLocation>
    </referencePoint>
    <centerLine>
        <LinearExtent gml:id="cl-C-10">
            <gml:posList srsDimension="3"
            srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
            uomLabels="degrees">39.475026 -81.795909 828.80 39.475026 -81.795909
813.77</gml:posList>
        </LinearExtent>
    </centerLine>
    <linearReferencing>
        <LinearSpatialReferenceSystem gml:id="lsr-C-10">
            <gml:identifier codeSpace="urn:x-def:authority:DIGGSINC">
                urn:x-diggs:def:fi:DIGGSINC:C-10-lsr</gml:identifier>
            <glr:linearElement xlink:href="#cl-C-10" />
            <glr:lrm>
                <glr:LinearReferencingMethod gml:id="lrm-C-10">
                    <glr:name>chainage</glr:name>
                    <glr:type>absolute</glr:type>
                    <glr:units>ft</glr:units>
                </glr:LinearReferencingMethod>
            </glr:lrm>
        </LinearSpatialReferenceSystem>
    </linearReferencing>
    <totalMeasuredDepth uom="ft">15.03</totalMeasuredDepth>
    <soundingPurpose>CPT</soundingPurpose>
</samplingFeature>
```

This is a lot all at once, but that is because it mirrors the same structure as our borehole for defining the location and geometry for our sounding. Instead of having a “boreholePurpose” though we have a “soundingPurpose” property.

With this, we have covered adding a samplingFeature for a sounding.

Associating Multiple Sampling Features

Often when working with boreholes or soundings, there is a desire to combine them into logical pairings or groups for reference. In the case of our code from the previous section, we want to create an association between our borehole and our CPT sounding sampling features. Neither is a parent of the other, so to accomplish this we'll be using a "group" or, more specifically, a "SamplingFeatureGroup." Let's look at the example:

```xml
<Diggs>
    <group>
        <SamplingFeatureGroup gml:id="sfg-1">
            <samplingFeatureRef xlink:href="#Location_B-001-0-20"/>
            <samplingFeatureRef xlink:href="#df-cpt_C-10"/>
        </SamplingFeatureGroup>
    </group>
</Diggs>
```

The "SamplingFeatureGroup" lends itself to being used in a pretty straightforward way. In this case, we just need to add two "samplingFeatureRef" properties with "xlink:href" attributes linking back to each corresponding sampling feature. The first is for our borehole and the second connects to our CPT sounding. Each "SamplingFeatureGroup" in DIGGS must have at least one "samplingFeatureRef" property, but this shouldn't be an issue in practice as you should always use two or more anyway.

Soil and Rock Descriptions

Introduction

This section of the guide covers documenting lithological information for our example borehole used in the borehole log based on the Ohio Department of Transportation (ODOT) which can be found here. Prior to working through this section of the guide, you should already know how to set up a DIGGS document. If this isn't already familiar to you, more information on setting up your DIGGS document can be found here.

The Lithology System

When it comes to the meat and potatoes of what does the ground look like where, the "LithologySystem" object is our work horse. Our Diggs root node can have any number of "observation" properties, and the "LithologySystem" object in this case will be its child. Let's get into some example code which we will break down. As always, some of the non-relevant XML has been abridged so that we can focus on what is relevant:

```xml
<Diggs>
    <observation>
        <LithologySystem gml:id="Litho_B-001-0-20">
            <projectRef xlink:href="#Project_600819" />
            <samplingFeatureRef xlink:href="#Location_B-001-0-20" />
            <lithologyClassificationType>ODOT</lithologyClassificationType>
        </LithologySystem>
    </observation>
</Diggs>
```

First and foremost, we want to connect this lithological information to the area that it is describing. We use the "projectRef" property with a "xlink:href" attribute to connect it to its parent project. With any "href" in DIGGS, we connect it by setting the attribute to the "gml:id" of the target object. This "LithologySystem" describes the layer composition of a particular "SamplingFeature," so we need to connect the two to denote that they are related. We do this in the same way as with connecting to a project through the "samplingFeatureRef" property.

Next, we have a required property "lithologyClassificationType." This is used to record which classification system is used for the soil layer composition which will follow this example. In this case, ODOT uses their own classification scheme that is not entirely unlike USCS, so it is denoted as, "ODOT."

Lithology Observations

Next, we need to record the actual data for our layers. We will be doing that with a series of "lithologyObservation" complex properties each with a "LithologyObservation" object. The topmost layer is a relatively simple one, so we'll start by looking at some asphalt.

```xml
<Diggs>
    <observation>
        <LithologySystem gml:id="Litho_B-001-0-20">
            <projectRef xlink:href="#Project_600819" />
            <samplingFeatureRef xlink:href="#Location_B-001-0-20" />
            <lithologyClassificationType>ODOT</lithologyClassificationType>
            <lithologyObservation>
                <LithologyObservation gml:id="Litho_Observation_B-001-0-20_0.0">
                    <location>
                        <LinearExtent gml:id="DGS4129-5FE-304D-B498-615C4">
                            <gml:posList srsName="#lsr-B-001-0-20"
                             srsDimension="1">0.0 1.5</gml:posList>
                        </LinearExtent>
                    </location>
                </LithologyObservation>
            </lithologyObservation>
        </LithologySystem>
    </observation>
</Diggs>
```

First, we need to define the extent of the layer in question. We do that with a “location” property with a “LinearExtent” object. The srsName attribute links back to the “LinearSpatialReferenceSystem” object which defines our units. More information on how to set up a “LinearSpatialReferenceSystem” can be found here. Secondly, we use the “srsDimension” attribute to set the number of dimensions in our shape. In this case, we are using a line to represent a portion of the borehole, so it is one dimensional. Finally, we have one row of two coordinates that represent the start and end of this first layer. So, we now know where but not what, so how do we say this layer is composed of asphalt? Let’s continue the example.

```
<lithologyObservation>
    <LithologyObservation gml:id="Litho_Observation_B-001-0-20_0.0">
        <location>
            <LinearExtent gml:id="DGS4129-5FE-304D-B498-615C4">
                <gml:posList srsName="#lsr-B-001-0-20"
                srsDimension="1">0.0 1.5</gml:posList>
            </LinearExtent>
        </location>
        <trueBaseObserved>true</trueBaseObserved>
        <primaryLithology>
            <Lithology gml:id="DGS100B-1519-46B5-2753-2D71D">
                <lithDescription>ASPHALT (18")</lithDescription>
                <legendCode>asphalt</legendCode>
            </Lithology>
        </primaryLithology>
    </LithologyObservation>
</lithologyObservation>
```

There are a couple of new pieces that have been added. First, there is now a “trueBaseObserved” simple property which is set to either true or false. This represents whether the bottom defined is the true bottom of the defined area. In this case, one and a half feet into the ground is indeed where the layer of asphalt ends as we’ve defined for this layer, so we set this to true. The second and more important piece of the “primaryLithology” property with a “Lithology” object. In this example, it has two children: “lithDescription” and “legendCode.”

The “lithDescription” property uses a string that describes the content of the layer. Because this first layer is asphalt, it has a pretty straightforward composition. The second property “legendCode” is used for report generation purposes, and it is intended to signal a corresponding graphic in the report. ODOT does not use a classification code for asphalt, so that is all we need for this topmost layer, but not all layers are this simple. Next, we will look at one that has more going on.

We will begin with what is the same. This piece of example code directly follows the one above. It also resides within a LithologySystem object:

```
<lithologyObservation>
    <LithologyObservation gml:id="Litho_Observation_B-001-0-20_1.5">
        <location>
            <LinearExtent gml:id="DGS4129-5FE-304D-B498-615C5">
                <gml:posList srsName="#lsr-B-001-0-20"
                srsDimension="1">1.5 4.5</gml:posList>
            </LinearExtent>
        </location>
    </LithologyObservation>
</lithologyObservation>
```

This location section works the same way as previously, but its top coordinate is the same as the previous layer's bottom coordinate. This is a data management best practice because layers are inherently continuous and your data should be describing the physical reality. This is what we expect, as it's directly below the preceding layer. So far so good. Let's look at how things begin to diverge:

```
<lithologyObservation>
    <LithologyObservation gml:id="Litho_Observation_B-001-0-20_1.5">
        <location>
            <LinearExtent gml:id="DGS4129-5FE-304D-B498-615C5">
                <gml:posList srsName="#lsr-B-001-0-20"
                srsDimension="1">1.5 4.5</gml:posList>
            </LinearExtent>
        </location>
        <primaryLithology>
            <Lithology gml:id="DGS100B-1519-46B5-2753-2D71C">
                <classificationCode codeSpace="USCS">SANDY
SILT</classificationCode>
                <classificationSymbol codeSpace="USCS">ML</classificationSymbol>
                <lithDescription>STIFF, REDDISH BROWN, SANDY SILT, SOME STONE
FRAGMENTS, LITTLE CLAY, DAMP
                </lithDescription>
                <legendCode>silt</legendCode>
                <matrixType>soil</matrixType>
            </Lithology>
        </primaryLithology>
    </LithologyObservation>
</lithologyObservation>
```

This time around, there are some more properties of the "Lithology" child of the "primaryLithology." Let's work through them. The "classificationCode" property references an authority that defines this code, in this case, "USCS" and has a value of SANDY SILT based on that standard. As mentioned previously, ODOT does not strictly adhere to the USCS standard, which is why the overall LithologySystem's "lithologyClassificationType" is set to, "ODOT" instead of, "USCS." Next, the "classificationSymbol" is used to record the value corresponding to the appropriate symbol.

Note from Elliotte: matrixType doesn't seem to appear in the documentation. Is this a new property?

Let's continue working through this example. Next up, we will further break down the characteristics of this layer into its constituents and a field property. But first, what color is it?

```xml
<primaryLithology>
    <Lithology gml:id="DGS100B-1519-46B5-2753-2D71C">
        <!-- Abridged XML -->
        <color>
            <Color gml:id="DGS4129-5CE-304D-B498-615C5" character="uniform">
                <colorName>Reddish brown</colorName>
            </Color>
        </color>
        <constituent>
            <Constituent gml:id="Litho_Observation_B-001-0-20_1.5_C1">
                <gml:name>Stone fragments</gml:name>
                <abundanceCode>some</abundanceCode>
            </Constituent>
        </constituent>
    </Lithology>
</primaryLithology>
```

It is reddish brown. We use the “color” property with an aptly named “Color” object. The “character” attribute is used to denote how the color appears or is distributed. In this case, it appears uniformly, but the complete list of possible values includes:

- banded
- mottled
- multicolored
- uniform
- variegated

There is only one property needed in this case which is “colorName” the value of which is set to, “Reddish brown,” a common coloration for clay.

Next up, we have a “constituent” property with a “Constituent” child object. This is one of two such properties within this “Lithology.” We use the “gml:name” attribute to describe the constituent, and the “abundanceCode” property to how abundant this constituent is within the lithology. In some cases, you might prefer to create a controlled list of possible values for “abundanceCode” in which case you would reference that authority using a “codeSpace” attribute.

For reference, the below contains the XML for a second constituent. It is much like the first, so we will not dwell on it, but instead talk about adding the “fieldProperties” property:

```xml
<primaryLithology>
    <Lithology gml:id="DGS100B-1519-46B5-2753-2D71C">
        <!-- Abridged XML -->
        <color>
            <Color gml:id="DGS4129-5CE-304D-B498-615C5" character="uniform">
                <colorName>Reddish brown</colorName>
            </Color>
        </color>
        <constituent>
            <Constituent gml:id="Litho_Observation_B-001-0-20_1.5_C1">
                <gml:name>Stone fragments</gml:name>
                <abundanceCode>some</abundanceCode>
            </Constituent>
        </constituent>
        <constituent>
            <Constituent gml:id="Litho_Observation_B-001-0-20_1.5_C2">
                <gml:name> Clay</gml:name>
                <abundanceCode>little</abundanceCode>
            </Constituent>
        </constituent>
        <fieldProperties>
            <FieldProperties gml:id="Litho_Observation_B-001-0-20_1.5_P1">
                <consistency>STIFF</consistency>
                <moistureCondition>DAMP</moistureCondition>
            </FieldProperties>
        </fieldProperties>
    </Lithology>
</primaryLithology>
```

With this latest addition, we are able to use the “consistency” and “moistureCondition” properties of the “FieldProperties” object to denote that the layer is both stiff and damp.

Now, we have the building blocks to continue in this manner to describe the remaining layers of the borehole. Subsequent layers are documented in DIGGS in the same manner as above.

The GeoUnitSystem

When working with soil layers, it’s often helpful to simplify the data to be able to better answer questions like, “Where do the layers of rock begin?” We can answer that question and questions like it using the “GeoUnitSystem” object. Let’s get into it by looking at an example,

```xml
<observation>
    <GeoUnitSystem gml:id="Strat_600819">
        <projectRef xlink:href="#Project_600819"/>
        <samplingFeatureRef xlink:href="#Location_B-001-0-20"/>
        <geoUnitType>geotechnical</geoUnitType>
    </GeoUnitSystem>
</observation>
```

The “GeoUnitSystem” object is housed within an observation property of the root, “Diggs” node. We associate it with its corresponding project and sampling feature through hrefs as we have seen in previous XML in the prior section. We also set the “geoUnitType” to “geotechnical” then we are ready to start adding a number of “geoUnitObservations.” These are more complicated objects that are used to indirectly group together a number of layers. Rather than adding these layers by reference, we will instead be using a “LinearExtent” object to define where this area begins and ends. Below is an example which we will break down:

```xml
<observation>
    <GeoUnitSystem gml:id="Strat_600819">
        <projectRef xlink:href="#Project_600819" />
        <samplingFeatureRef xlink:href="#Location_B-001-0-20" />
        <geoUnitType>geotechnical</geoUnitType>
        <geoUnitObservation>
            <GeoUnitObservation gml:id="Strat_600819_01">
                <location>
                    <LinearExtent gml:id="Strat_600819_01_loc">
                        <gml:posList srsName="#lsr-B-001-0-20"
srsDimension="1">0.0 15.0</gml:posList>
                    </LinearExtent>
                </location>
                <unitName>Soil</unitName>
                <verticalDistanceToBaseOfUnit
uom="ft">15.0</verticalDistanceToBaseOfUnit>
            </GeoUnitObservation>
        </geoUnitObservation>
</observation>
```

Each “GeoUnitObservation” object is housed sequentially from top to bottom in its own “geoUnitObservation” property. We use a location property with a “LinearExtent” object to define the portion of the borehole corresponding to this area; finally we define the distance vertically to the base of this area with the “verticalDistanceToBaseOfUnit” property. Unlike with the “LinearExtent” object previously, we don’t have units defined elsewhere for our “verticalDistanceToBaseOfUnit,” so we explicitly define that it is in feet by setting the “uom” property to, “ft.”

There is a second GeoUnitObservation following this one that works in the same way with one key difference. We want to know where the rock starts, so instead of using the vertical distance to the base of this area of the borehole, we want to know the vertical distance to the top of it. This is what that looks like:

```
<geoUnitObservation>
    <GeoUnitObservation gml:id="Strat_600819_02">
        <location>
            <LinearExtent gml:id="Strat_600819_02_loc">
                <gml:posList srsName="#lsr-B-001-0-20" srsDimension="1">15.0
41.0</gml:posList>
            </LinearExtent>
        </location>
        <unitName>Rock</unitName>
        <verticalDistanceToTopOfUnit uom="ft">15.0</verticalDistanceToTopOfUnit>
    </GeoUnitObservation>
</geoUnitObservation>
```

It is very similar, but the property we want is instead "verticalDistanceToTopOfUnit."

That is it for Soil and Rock Descriptions. If you are following along with this guide in order, next up is SPT and Rock Core Samples for a Borehole. We'll see you over there.

Borehole SPT and Rock Core Sampling

Introduction

In previous guides, we have looked at sampling features, but now we will be looking at other aspects of the sampling process, sampling activities and samples. Using these two DIGGS objects, we can define how samples were produced and the traits of the sample itself. We will not, however, be looking into testing performed on these produced samples. Information on recording testing can instead be found here.

The data for this guide will be derived from an example borehole log based on the Ohio Department of Transportation (ODOT) which can be found here. In this guide, we will be using ODOT data to record how an SPT sample was retrieved.

Sampling Activity

So, what is a sampling activity? It is the action taken to retrieve a sample, regardless of whether a sample was successfully retrieved. Usually these are associated with a location on a sampling feature, but can also occur elsewhere such as in a lab if there are test or blank samples, or if there are aggregate samples without a connection to a specific location. In other words, it is the action of creating or attempting to create a sample.

We will begin by looking at some high levels properties we need to get things set up.

```xml
<samplingActivity>
    <SamplingActivity gml:id="SPT_SA_B-001-0-20_1.5_SS">
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819"/>
        <samplingFeatureRef xlink:href="#Location_B-001-0-20"/>
    </SamplingActivity>
</samplingActivity>
```

Each sampling activity lives within a series of “samplingActivity” properties within the “Diggs” root node. Each of these has a single child, a “SamplingActivity” object. If you are unclear why we have an XML node with almost the exact same name nested inside of another, you should read about the “Object-Property Rule” explained in the Initializing a DIGGS Document and Creating a DIGGS Project document. Pressing on, the first property is “investigationTarget” which uses a string to indicate the target of the investigation, but the list of possible values is restricted to one of the following:

- Deep Foundation - investigation of deep foundation elements
- Earthworks -earth materials excavated and/or emplaced by engineering activity
- Ground Water -hydrology and chemistry of ground water
- Indoor Atmosphere - meteorology and chemistry of air in indoor spaces
- Natural Ground - earth materials sampled in natural or undisturbed state
- Outdoor Atmosphere - meteorology and chemistry of air in outdoor spaces
- Surface Water - hydrology and chemistry of surface water

In this case, “Natural Ground” is appropriate for this sampling activity as we are gathering it from ground adjacent to a highway.

For the next two properties, we want to associate the sampling activity with their appropriate project and sampling feature. We do this with the “projectRef” and “samplingFeatureRef” properties respectively, each of which uses an “xlink:href” attribute with the “gml:id” of for the project and sampling feature preceded by a, “#” character.

Next, let’s look at the dimensions of where we retrieved the sample.

```xml
<samplingActivity>
    <SamplingActivity gml:id="SPT_SA_B-001-0-20_1.5_SS">
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819"/>
        <samplingFeatureRef xlink:href="#Location_B-001-0-20"/>
        <samplingLocation>
            <LinearExtent gml:id="DGSA22F-342-D3A-EBE9-489F2" srsDimension="1"
                srsName="#lsr-B-001-0-20">
                <gml:posList>1.5 3.00</gml:posList>
            </LinearExtent>
        </samplingLocation>
        <activityType>collect</activityType>
    </SamplingActivity>
</samplingActivity>
```

We now have a “samplingLocation” property with a one dimensional “LinearExtent” defining the portion of the borehole from which the same was retrieved as between 1.5 and 3.00 feet. No unit of measure is provided in the above code snippet because it is defined in the referenced “LinearSpatialReferenceSystem” which has a “gml:id” of, “lsr-B-001-0-20.” More information on how this works can be found here.

As for the “activityType,” this refers to the nature of how a sample was produced. In our case, it was collected directly from a borehole, but there are other methods for actions such as samples produced by combining two or more other samples or samples produced by conducting a test. Depending on the method of producing the sample, it may be necessary to add one or more “sourceSampleRef” properties with an “xlink:href” attribute pointing to the relevant samples. In this case, one is not required as our sample was collected directly from a sampling feature (our borehole).

- Collect -activity created a sample or samples by collection from a sampling feature;
- Aggregate - activity created a sample or samples by aggregating existing samples; the associated Sample(s) should contain more than one sourceSampleRef;
- Subsample - activity created a sample or samples by subsampling an existing sample; only one “sourceSampleRef” should be specified for the related Sample(s);

- Test- activity produced a test, standard or blank sample that does not relate to any field sample or sampling feature; activity will point to a project and no "sourceSampleRef" should be specified for the associated Sample(s).
- None - the sample activity failed to produce any physical sample;
- Unknown - used where no information about the sampling activity exists (eg. for legacy data where only sample name or label and location (depth) exists).

With that we are caught up with our example code, so let's break down the rest starting with the "sampleProduced" property:

```xml
<samplingActivity>
    <SamplingActivity gml:id="SPT_SA_B-001-0-20_1.5_SS">
        <!-- Abridged XML -->
        <activityType>collect</activityType>
        <sampleProduced>
            <SampleProduced gml:id="SampleProduced_B-001-0-20_1.50_3.00_SS">
                <location>
                    <LinearExtent gml:id="DGS1D76-F2D-E40-97D2-4ED80"
                        srsDimension="1"
                        srsName="#lsr-B-001-0-20">
                        <gml:posList>1.50 2.58</gml:posList>
                    </LinearExtent>
                </location>
            </SampleProduced>
        </sampleProduced>
    </SamplingActivity>
</samplingActivity>
```

It has one child, "SampleProduced." It is important to note that this "SampleProduced" object will be referenced later by a corresponding sample, so take note of the "gml:id" we select here. That being said in our case we need just one property for, a "location" with a "linearExtent" defining the portion of the area successfully collected. A bit later, we are going to notate that the sample was 1.08 feet in length, so we defined this extent by adding that length to the top depth of the "linearExtent" from which the sample was taken. We don't need to define the units of measure for the same reason as before.

Let's continue:

```xml
<samplingActivity>
    <SamplingActivity gml:id="SPT_SA_B-001-0-20_1.5_SS">
        <!-- Abridged XML -->
        <activityType>collect</activityType>
        <sampleProduced>
            <SampleProduced gml:id="SampleProduced_B-001-0-20_1.50_3.00_SS">
                <location>
                    <LinearExtent gml:id="DGS1D76-F2D-E40-97D2-4ED80"
srsDimension="1" srsName="#lsr-B-001-0-20">
                        <gml:posList>1.50 2.58</gml:posList>
                    </LinearExtent>
                </location>
            </SampleProduced>
        </sampleProduced>
        <samplerInsertionMethod>driven</samplerInsertionMethod>
        <samplingMethod xlink:href="#DGS490A-782-2E22-1768-42C85" />
        <totalSampleRecoveryLength uom="ft">1.08</totalSampleRecoveryLength>
    </SamplingActivity>
</samplingActivity>
```

First up, we've added a "samplerInsertionmethod" property which is set to, "driven." This property is used when sampling soil or rock to define the means by which the sampler is inserted into the ground. The value of this property should come from a controlled list which might have values such as pushed or driven.

Next, the "samplingMethod" links to a corresponding "procedure." The object is a "diggs_geo:DrivenPenetrationTest" in this case. Explaining this linked object in depth is beyond the scope of this section of the guide, but it is covered here. For now, it is enough to understand that it links to elsewhere which provides more information about the method by which the sample was obtained.

Finally, the "totalSampleRecoverLength" defines the length of the recovered material 1.08 feet. The length is recorded as the value, and the unit of measure is defined in the "uom" property.

Sample

In DIGGS, what is a sample? Well, it is the physical sample (earth material, fluid, gas) obtained from the sampling activity for observation or testing. All material samples must be associated with a sampling activity. Samples are collected from sampling features as a result of sampling activities. Samples can also be produced by sampling activities via aggregation, subsampling, or creating a test standard in a lab. All material samples must refer to a sampling activity.

The sample, sampling activity, and sampling feature are broken apart so that portions of data can be transmitted without having to share where a sample came from or its expected characteristics. We want to be able to transmit certain data to labs without biasing them (important for geoenvironmental applicatons). Due to that, as we break down this final third of the sample triangle, you will notice there is some redundant information. This is an intended feature.

Let's look at how to set up our sample in DIGGS. As before, this sample is nested within the "Diggs" root node:

```
<sample>
    <Sample gml:id="Sample_B-001-0-20_1.50_1_SS_">
        <gml:description>Reddish brown sandy silt, A-4a (1)</gml:description>
        <gml:name>SS-1</gml:name>
        <projectRef xlink:href="#Project_600819"/>
        <samplingActivityRef xlink:href="#SPT_SA_B-001-0-20_1.5_SS"/>
        <sampleProducedRef xlink:href="#SampleProduced_B-001-0-20_1.50_3.00_SS"/>
    </Sample>
</sample>
```

Here we have a "sample" property with a "Sample" object as its only child. Within that, we have a number of familiar properties. The "gml:description" and "gml:name" describe the sample and provide what should ideally be a unique identifying name within the project. Speaking of the project, we reference it with a "projectRef" and similarly are connecting to the prior "samplingActivity" with the "samplingActivityRef" property. The main new curve ball here is the "sampleProducedRef" property. Previously, we mentioned that the sampling activity's sample produced would be linked to, and this is where that happens. We take that "gml:id" and plug it in here.

Let's continue:

```
<sample>
    <Sample gml:id="Sample_B-001-0-20_1.50_1_SS_">
        <gml:description>Reddish brown sandy silt, A-4a (1)</gml:description>
        <gml:name>SS-1</gml:name>
        <projectRef xlink:href="#Project_600819"/>
        <samplingActivityRef xlink:href="#SPT_SA_B-001-0-20_1.5_SS"/>
        <sampleProducedRef xlink:href="#SampleProduced_B-001-0-20_1.50_3.00_SS"/>
        <classification>Soil</classification>
        <condition>disturbed</condition>
        <sampleDimensions>
            <SampleDimensions>
                <length uom="ft">1.08</length>
            </SampleDimensions>
        </sampleDimensions>
    </Sample>
</sample>
```

Firstly, we have added the "classification" property with a value of "Soil;" this is used to define the class of sample collected which ideally should come from a controlled list containing possible values such as soil, rock or fluid. Similarly, we record the condition of the sample with the "condition" property, but in this case the string is not controlled.

Finally, we record the size of the sample with the “sampleDimensions” property and corresponding object. Because this sample is linear in nature, a “length” property is appropriate for defining its size, “1.08” in feet as recorded by the “uom” attribute. As noted previously, this information is redundant to what is recorded elsewhere, but it is placed here additionally to allow for the severability of this data.

Next Steps

With that, we’ve covered the final two pillars of the sampling trifecta, but what if we want to record the results we obtained from tests performed on these samples? To answer that question, you should continue onto the Lab Results for SPT Soil and Rock Samples Obtained from Borehole guide coming up next.

Recording SPT, CPT, and Atterberg Limit Test Results in DIGGS

Introduction

This section of the guide covers how to record test results which will be exemplified with data from SPT, CPT, and Atterberg Limit tests. This example data is taken from some mock results from the Ohio Department of Transportation; it is extracted from a borehole log which can be found here.

Prior to working through this section of the guide, you should already know how to set up a DIGGS document. If this is not already familiar to you, more information on setting up your DIGGS document can be found here.

In the following code samples, it is often necessary to abridge large portions of the code to improve the readability of a particular section that is being highlighted. If you find yourself struggling to figure out the order of how pieces fit together, you can reference the full unabridged code for this article here.

SPT Tests

To record our SPT test results, we will be working with the "Test" object. This object resides within a "measurement" property directly within the "Diggs" root object. There can and often will be multiple "measurement" properties with a "Test" object. To begin, let's look at some example code that we will use to identify our SPT test and associate it with its project and sampling feature:

```
<measurement>
    <Test gml:id="SPT_B-001-0-20_1.5">
        <gml:name>SPT</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819"/>
        <samplingFeatureRef xlink:href="#Location_B-001-0-20"/>
    </Test>
</measurement>
```

So, here we have our "Test" object that we are using for recording an SPT test. We use the "gml:name" attribute to make this immediately recognizable at a glance by setting the value to, "SPT."

The "investigationTarget" property uses a string to indicate the target of the investigation, but the list of possible values is restricted to one of the following:

- Deep Foundation - investigation of deep foundation elements
- Earthworks -earth materials excavated and/or emplaced by engineering activity
- Ground Water -hydrology and chemistry of ground water;
- Indoor Atmosphere - meteorology and chemistry of air in indoor spaces
- Natural Ground - earth materials sampled in natural or undisturbed state;
- Outdoor Atmosphere - meteorology and chemistry of air in outdoor spaces
- Surface Water - hydrology and chemistry of surface water;

We then reference the corresponding project and sampling feature with the "projectRef" and "samplingFeatureRef" properties respectively. Each uses an "xlink:href" attribute to connect to their corresponding element. The key is to use the “gml:id” of the object being referenced as the value of this attribute pre-appended with a, “#” character.

With these basic properties set up, we are ready to get into the actual data. We will begin by defining the section of the borehole sampled.

```
<measurement>
    <Test gml:id="SPT_B-001-0-20_1.5">
        <gml:name>SPT</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
        <samplingFeatureRef xlink:href="#Location_B-001-0-20" />
    </Test>
    <outcome>
        <TestResult gml:id="DGSA78B-2FE-1942-FCD9-20E22">
            <location>
                <LinearExtent srsDimension="1" srsName="lsr-B-001-0-20"
                    gml:id="DGS7979-243-BAD-B407-1A38B">
                    <gml:posList>1.5 3.00</gml:posList>
                </LinearExtent>
            </location>
        </TestResult>
    </outcome>
</measurement>
```

The “location” property is used to define this area where the sample was taken using a “LinearExtent” object. How this object works has been covered previously, and as such you can find more information on it here.

Next up, we are going to define the columns for our result data:

```
<TestResult gml:id="DGSA78B-2FE-1942-FCD9-20E22">
    <location>...</location>
    <results>
        <ResultSet>
            <parameters>
                <PropertyParameters gml:id="DGS68A7-B7E-19B4-18DC-1CED4">
                    <properties>
                        <Property index="1" gml:id="DGS1A35-BC5-C26-F243-5944D">
                            <propertyName>N-Value</propertyName>
                            <typeData>long</typeData>
                            <propertyClass
                                codeSpace="http://diggsml.org/dictionar-
ies/DIGGSTestPropertyDefinitions.xml">
                                n_value</propertyClass>
                        </Property>
                    </properties>
                </PropertyParameters>
            </parameters>
        </ResultSet>
    </results>
</TestResult>
```

This will take place within the “results” parameter which contains a “ResultSet” object. We define which columns of data are in the result set by listing a number of “Property” objects within the “properties” property of our “PropertyParameters.” This structure may seem like overkill now with only one parameter, but the need for the structure will become more apparent as more we use it to define more columns then after add the actual results. For now, let’s look at how to define a “Property” object.

Each “Property” has an index attribute which lists the order in which it appears. Start with one then increment by one for each subsequent “Property” object in the collection without repeating any values. Now, let’s look at the properties of our “Property” object: “propertyName,” “typeData,” and “propertyClass.”

The “propertyName” records a human readable name for the property and is often used as a column header in reports. We use “typeData” to refer to the data type of the data which is in this column. This may be any number of common data types such as a float, string, integer, or as is the case here, “long.” As for the “propertyClass,” this is used enter the gml:id for the N-Value measurement defined in the corresponding dictionary which is referenced using the “codeSpace” attribute. In this case, we are using a standard dictionary that is available with many measurements standard to the industry. Let’s look at this excerpt from the dictionary before we return to the XML we have been working on.

```
<dictionaryEntry>
    <Definition gml:id="n_value">
        <gml:description>
            Value from Standard Penetration Test (SPT) to define advancement in
terms of blows required to drive a specific sampler I foot into the soil using a
hammer that provides a specific impact energy.
        </gml:description>
        <gml:identifier codeSpace="http://diggsml.org/terms"> N-value (raw)
</gml:identifier>
        <dataType>integer</dataType>
        <uomType>dimensionless</uomType>
        <authority />
        <associatedElement>
            /diggs:Diggs/diggs:measurement/diggs:Test/diggs:procedure/diggs_geo:D
rivenPenetrationTest
        </associatedElement>
    </Definition>
</dictionaryEntry>
```

As you can see, it defines the aspects of this measurement, and we used the “gml:id” value of the “Definition” to reference it. There are a number of other properties for this dictionary entry that help to define it which can be perused above. If you would like to see the other values in the dictionary, the URL referenced in the above “codeSpace” attribute is live and can be viewed here.

Returning to where we were before, we will now follow the same process to add a second column with another measurement:

```
<PropertyParameters gml:id="DGS68A7-B7E-19B4-18DC-1CED4">
    <properties>
        <Property index="1" gml:id="DGS1A35-BC5-C26-F243-5944D">
            <propertyName>N-Value</propertyName>
            <typeData>long</typeData>
            <propertyClass
                codeSpace="http://diggsml.org/dictionaries/DIGGSTestPropertyDefin
itions.xml">
                n_value</propertyClass>
        </Property>
        <Property index="2" gml:id="DGS1A35-BC5-C26-F243-5944E">
            <propertyName>N60</propertyName>
            <typeData>long</typeData>
            <propertyClass
                codeSpace="http://diggsml.org/dictionaries/DIGGSTestPropertyDefin
itions.xml">
                n1_60</propertyClass>
        </Property>
    </properties>
</PropertyParameters>
```

This is largely the same, but it is for the N-60 measurement. Note how the “index” attribute of the “Property” object has iterated between the first and second entry.

Alright, now we will add the actual data. In this case. We have already done the work to define what measurements will appear in which order, so the data itself may seem somewhat minimal in this case. That is by design. Let’s look at the code:

```xml
<ResultSet>
    <parameters>
        <PropertyParameters gml:id="DGS68A7-B7E-19B4-18DC-1CED4">
            <properties>
                <Property index="1" gml:id="DGS1A35-BC5-C26-F243-5944D">
                    <propertyName>N-Value</propertyName>
                    <typeData>long</typeData>
                    <propertyClass codeSpace="http://diggsml.org/dictionar-
ies/DIGGSTestPropertyDefinitions.xml">n_value</propertyClass>
                </Property>
                <Property index="2" gml:id="DGS1A35-BC5-C26-F243-5944E">
                    <propertyName>N60</propertyName>
                    <typeData>long</typeData>
                    <propertyClass codeSpace="http://diggsml.org/dictionar-
ies/DIGGSTestPropertyDefinitions.xml">n1_60</propertyClass>
                </Property>
            </properties>
        </PropertyParameters>
    </parameters>
    <dataValues cs="," ts=" " decimal=".">17,24</dataValues>
</ResultSet>
```

It is only one line! We only need to record one N-Value measurement and one N-60 measurement, so we only have a single row of data. To disambiguate data shared in differing cultural contests, we use the "cs" property to define the comma separator which appears to separate data values within a row and the "decimal" property is used to define if a comma or period is used for as the decimal character.

That is it for our "outcome" property. But how did things go when we were actually collecting that data? How many blow counts were there for each interval and what type of test was it ? We are going to record this information using the "procedure" property which is right after the "outcome" one we just finished.

In this case, our procedure's information is entered by adding a "diggs_geo:DrivenPenetrationTest" object. Let's begin:

```
<procedure>
    <diggs_geo:DrivenPenetrationTest gml:id="DGS490A-782-2E22-1768-42C85">
        <testProcedureMethod>
            <Specification gml:id="ASTM_D1586_D1586M-18e1">
                <gml:name>Standard Test Method for Standard Penetration Test
(SPT) and Split-BarrelSampling of Soils</gml:name>
                <accreditingBody>ASTM</accredtingBody>
                <shortMethodName>ASTM 1586/1586M</shortMethodName>
            </Specification>
        </testProcedureMethod>
        <diggs_geo:penetrationTestType>SPT</diggs_geo:penetrationTestType>
    </diggs_geo:DrivenPenetrationTest>
</procedure>
```

First, we add a “testProcedureMethod” property with a “Specification” object as a child. This specification has three properties that we will need for recording our SPT test. The “gml:name” provides a human readable and easily identifiable description of the test in question. We use the “accreditingBody” property to refer to the organization in charge of this specification. In our case, this is the ASTM standard. Finally, we use the “shortMethodName” property to refer to the specific method name or number as they have defined it.

Pulling back a bit in the schema, we record that this is an SPT test in the “diggs_geo:penetrationTestType” property.

Next up, we will look at a few more important properties for recording the set-up for the SPT testing:

```
<procedure>
    <diggs_geo:DrivenPenetrationTest gml:id="DGS490A-782-2E22-1768-42C85">
        <diggs_geo:penetrationTestType>SPT</diggs_geo:penetrationTestType>
        <diggs_geo:hammerType>CME Automatic</diggs_geo:hammerType>
        <diggs_geo:hammerEfficiency uom="%">84</diggs_geo:hammerEfficiency>
        <diggs_geo:totalPenetration uom="ft">1.50</diggs_geo:totalPenetration>
    </diggs_geo:DrivenPenetrationTest>
</procedure>
```

So, we have three more properties now “diggs_geo:hammerType,” “diggs_geo:hammerEfficiency,” and “diggs_geo:totalPenetration.” In this example case from ODOT, a CME Automatic hammer was used at 84% efficiency to achieve a total penetration of one and a half feet. We need to set the “uom” attributes of “diggs_geo:hammerEfficiency” and “diggs_geo:totalPenetration” accordingly to test the units, then the actual values are just the corresponding numbers.

Finally, the “procedure” is closed out by a number of “diggs_geo:driveSet” objects that describe how the performance of the SPT test went. Let’s look at a snippet off just the first one:

```
<diggs_geo:driveSet>
    <diggs_geo:DriveSet>
        <diggs_geo:index>1</diggs_geo:index>
        <diggs_geo:blowCount>9</diggs_geo:blowCount>
        <diggs_geo:penetration uom="ft">0.5</diggs_geo:penetration>
    </diggs_geo:DriveSet>
</diggs_geo:driveSet>
```

The order of each “diggs_geo:DriveSet” is recorded sequentially using the “diggs_geo:index” property starting with one then incrementing by one each time. The blow count is recorded in the “diggs_geo:blowCount” property, and the amount of penetration is recorded in the “diggs_geo:penetration” object. Each time a “uom” attribute needs to be used for the depth.

This process is repeated a number of times until all the data from the SPT is recorded. By the end, it looks like this:

```
<procedure>
    <diggs_geo:DrivenPenetrationTest gml:id="DGS490A-782-2E22-1768-42C85">
        <testProcedureMethod>
            <Specification gml:id="ASTM_D1586_D1586M-18e1">
                <gml:name>Standard Test Method for Standard Penetration Test
(SPT) and Split-Barrel
                    Sampling of Soils</gml:name>
                <accredtingBody>ASTM</accredtingBody>
                <shortMethodName>ASTM 1586/1586M</shortMethodName>
            </Specification>
        </testProcedureMethod>
        <diggs_geo:penetrationTestType>SPT</diggs_geo:penetrationTestType>
        <diggs_geo:hammerType>CME Automatic</diggs_geo:hammerType>
        <diggs_geo:hammerEfficiency uom="%">84</diggs_geo:hammerEfficiency>
        <diggs_geo:totalPenetration uom="ft">1.50</diggs_geo:totalPenetration>
<diggs_geo:driveSet>
    <diggs_geo:DriveSet>
        <diggs_geo:index>1</diggs_geo:index>
        <diggs_geo:blowCount>9</diggs_geo:blowCount>
        <diggs_geo:penetration uom="ft">0.5</diggs_geo:penetration>
    </diggs_geo:DriveSet>
</diggs_geo:driveSet>
        <diggs_geo:driveSet>
            <diggs_geo:DriveSet>
                <diggs_geo:index>2</diggs_geo:index>
                <diggs_geo:blowCount>8</diggs_geo:blowCount>
                <diggs_geo:penetration uom="ft">0.5</diggs_geo:penetration>
            </diggs_geo:DriveSet>
        </diggs_geo:driveSet>
        <diggs_geo:driveSet>
            <diggs_geo:DriveSet>
                <diggs_geo:index>3</diggs_geo:index>
                <diggs_geo:blowCount>9</diggs_geo:blowCount>
                <diggs_geo:penetration uom="ft">0.5</diggs_geo:penetration>
            </diggs_geo:DriveSet>
        </diggs_geo:driveSet>
    </diggs_geo:DrivenPenetrationTest>
</procedure>
```

With that, we have covered using a “Test” object to record SPT data, but what about for a CPT sounding? The process is quite similar.

CPT Soundings

The beginning is largely the same. We record the name, investigation target, and reference the relevant project and sampling feature in the same manner as before:

```xml
<measurement>
    <Test gml:id="CPT_C-10">
        <gml:name>CPT Sounding</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
        <samplingFeatureRef xlink:href="#df-cpt_C-10" />
    </Test>
</measurement>
```

There are minor differences in the data as we are referencing a sounding instead of a borehole, but the structure is the same. Next we will add the beginning of the “outcome” property. It works in the same way as before except its location is different. Because it is so large, we will look at just the “MultiPointLocation” object which resides within the “location” property of the “outcome.”

```xml
<MultiPointLocation srsName="#cptsr1" srsDimension="1"
    gml:id="DGS159E-F10-2804-1446-57BC0">
    <gml:posList> 0.1530 0.2110 0.2920 0.3460 0.4260 0.4780 0.5660 0.6220
        0.6940 0.7670 0.8220 0.8690 0.9360 1.0010 1.0450 1.0990 1.1350
        1.1940 1.2490 1.2890 1.3470 1.3910 1.4420 1.4970 1.5540 1.5940
        1.6510 1.7050 1.8720 1.9100 1.9660 2.0220 2.0720 2.1260 2.1620
        2.2230 2.2580 2.3160 2.3680 2.4040 2.4640 2.5000 2.5570 2.6100
        2.6460 2.7080 2.7590 2.8020 2.8580 2.8950 2.9550 3.0040 3.0540
        3.1100 3.1520 3.2080 3.2650 3.3060 3.3570 3.4100 3.4610 3.5160
        3.5730 3.6110 3.6680 3.7200 3.7760 3.8330 3.8710 3.9360 3.9740
        4.0370 4.0940 4.1340 4.2010 4.2550 4.3070 4.3600 4.3990 4.4570
        4.5090 4.5680 4.6270 4.6650 4.7210 4.7740 4.8280 4.8800 4.9360
        4.9890 5.0470 5.2340 5.2980 5.3630 5.4050 5.4690 5.5270 5.5890
        5.6520 5.6940 5.7580 5.8140 5.8700 5.9270 5.9870 6.0440 6.1050
        6.1700 6.2120 6.2750 6.3330 6.3940 6.4570 6.5000 6.5650 6.6220
        6.6830 6.7470 6.7900 6.8530 6.9100 6.9730 7.0380 7.0790 7.1430
        7.2000 7.2620 7.3260 7.3690 7.4350 7.4930 7.5490 7.6080 7.6710
        7.7290 7.7930 7.8600 7.9070 7.9770 8.0210 8.0930 8.1590 8.2040
        8.2810 8.4670 8.5480 8.5950 8.6710 8.7420 8.7890 8.8690 8.9150
        8.9920 9.0620 9.1090 9.1890 9.2360 9.3110 9.3820 9.4300 9.5080
        9.5550 9.6310 9.7010 9.7470 9.8170 9.8800 9.9480 10.0150 10.0590
        10.1210 10.1820 10.2450 10.2860 10.3450 10.4050 10.4590 10.5180
        10.5580 10.6160 10.6690 10.7290 10.7860 10.8260 10.8850 10.9380
        10.9940 11.0530 11.0910 11.1490 11.1990 11.2530 11.3090 11.3470
        11.4030 11.4530 11.5080 11.5640 11.6970 11.7520 11.7890 11.8520
        11.8890 11.9480 12.0030 12.0400 12.0940 12.1430 12.1980 12.2420
        12.3020 12.3380 12.3980 12.4470 12.5030 12.5400 12.5980 12.6470
        12.6970 12.7530 12.7880 12.8470 12.8960 12.9510 12.9870 13.0400
        13.0950 13.1450 13.1990 13.2340 13.2960 13.3330 13.3920 13.4360
        13.4950 13.5320 13.5870 13.6400 13.6930 13.7300 13.7910 13.8420
        13.8850 13.9400 13.9770 14.0360 14.0830 14.1320 14.1860 14.2230
        14.2760 14.3240 14.3750 14.4250 14.4830 14.5180 14.5760 14.6310
        14.6680 14.7280 14.7650 14.8240 15.0340 </gml:posList>
</MultiPointLocation>
```

This is very large! Instead of a simple linear extent, we use this multi point location object so we can record a very large number of points of data we recorded data for along this line. Aside from this large swathe of data, it otherwise is quite similar to the linear extent encountered previously.

Next, we will look at how we define the columns for this CPT test:

```xml
<results>
    <ResultSet>
        <parameters>
            <PropertyParameters gml:id="pp_DGS808D-80E-43A1-2933-121A4">
                <properties>
                    <Property gml:id="qc_DGS20F1-3CA-34EF-B238-515D5" index="1">
                        <propertyName>qc</propertyName>
                        <typeData>double</typeData>
                        <propertyClass
                            codeSpace="http://diggsml.org/dictionaries/DIGGSTestP
ropertyDefinitions.xml">
                            tip_resistance</propertyClass>
                        <uom>tonf[US]/ft2</uom>
                        <nullValue reason="missing">9999</nullValue>
                    </Property>
                    <Property gml:id="fs_DGS1A70-1055-86-67A0-5B836" index="2">
                        <propertyName>fs</propertyName>
                        <typeData>double</typeData>
                        <propertyClass
                            codeSpace="http://diggsml.org/dictionaries/DIGGSTestP
ropertyDefinitions.xml">
                            sleeve_friction</propertyClass>
                        <uom>tonf[US]/ft2</uom>
                        <nullValue reason="missing">9999</nullValue>
                    </Property>
                    <Property gml:id="u2_DGS985E-526-4ED-722C-55823" index="3">
                        <propertyName>u2</propertyName>
                        <typeData>double</typeData>
                        <propertyClass
                            codeSpace="http://diggsml.org/dictionaries/DIGGSTestP
ropertyDefinitions.xml">
                            pore_water_pressure</propertyClass>
                        <uom>tonf[US]/ft2</uom>
                        <nullValue reason="missing">9999</nullValue>
                    </Property>
                </properties>
            </PropertyParameters>
        </parameters>
    </ResultSet>
</results>
```

The measurements themselves are different, but they work just as before with the “Property” objects for defining SPT measurements. In this case however, there is one key difference, we have added a “nullValue” property to each. The “nullValue” property is used to define which value is recorded in the event of a null result. We use the “reason” attribute to define why a null result may be present in the data, which in this case is “missing” to denote that datum wasn’t recorded. Because the value of this is set to, “9999” any such results in this column are instead construed to mean that particular datum is missing. We will close out the “results” in the same manner as before with the “dataValues” object. Each row corresponds to a different point recorded in the “poslist” from before, so there are quite a lot of them. The abridged data can be seen below:

```
<results>
    <ResultSet>
        <parameters>
            <PropertyParameters gml:id="pp_DGS808D-80E-43A1-2933-121A4">
                <properties>
                    <...>
                </properties>
            </PropertyParameters>
        </parameters>
        <dataValues cs="," decimal="." ts=" ">
            16.100,0.0000,0.0600
            26.200,0.0000,0.0600
            41.800,0.3500,0.0700
            <...>
            81.100,0.4800,0.2200
            82.900,0.4800,0.2300
            84.100,0.4700,0.2200
            84.600,0.0000,0.2300
            78.100,0.0000,0.2300
        </dataValues>
    </ResultSet>
</results>
```

As we defined three measurements using “Property” objects instead of two, we now have three columns of comma separated data. We can refer back to the index of each “Property” to see which is being referenced here, but as a convention having them appear in order helps to also reinforce the relationship implicitly.

Next up, we will leave the “outcome” and get into the “procedure.” We begin much in the same was as previously, except the “procedure” property houses a “diggs_geo:StaticConePenetrationTest” object instead as it’s for CPT. The structure of it still has a number of similarities though, starting with its “testProcedureMethod” as below:

```
<procedure>
    <diggs_geo:StaticConePenetrationTest gml:id="cpt_DGS88D1-BC6-3E5F-7E94-
68907">
        <testProcedureMethod>
            <Specification gml:id="ASTM_D3441-16">
                <gml:name>Standard Test Method for Mechanical Cone Penetration
Testing of Soils</gml:name>
                <accredtingBody>ASTM</accredtingBody>
                <shortMethodName>ASTM_D3441</shortMethodName>
            </Specification>
        </testProcedureMethod>
    </diggs_geo:StaticConePenetrationTest>
</procedure>
```

The test procedure method works in the same manner as before; it merely has different values as this is for CPT. Next up, we add the cone we used with the "testProcedureEquipment" property with an "Equipment" object:

```
<procedure>
    <diggs_geo:StaticConePenetrationTest gml:id="cpt_DGS88D1-BC6-3E5F-7E94-
68907">
        <testProcedureMethod>
            <Specification gml:id="ASTM_D3441-16">
                <gml:name>Standard Test Method for Mechanical Cone Penetration
Testing of Soils</gml:name>
                <accredtingBody>ASTM</accredtingBody>
                <shortMethodName>ASTM_D3441</shortMethodName>
            </Specification>
        </testProcedureMethod>
        <testProcedureEquipment>
            <Equipment gml:id="coneID_DGS4F57-821-2066-F520-4F31E">
                <gml:name>Cone</gml:name>
                <serialNumber>128.074</serialNumber>
            </Equipment>
        </testProcedureEquipment>
    </diggs_geo:StaticConePenetrationTest>
</procedure>
```

We need to use only two of its properties. We use the "gml:name" to define that it is a cone, and the "serialNumber" property allows us to have traceability to the specific piece of equipment.

For the purposes of making an illustrative example, we will next look at working with an assumed water depth as well as recording striking water as an event. First, let's look at assumed water depth:

```
<procedure>
    <diggs_geo:StaticConePenetrationTest gml:id="cpt_DGS88D1-BC6-3E5F-7E94-
68907">
        <otherTestProperty>
            <Parameter gml:id="wd_DGS1AE6-17D-3911-A256-272FA">
                <parameterName>Assumed Water Depth</parameterName>
                <parameterValue>6.00</parameterValue>
                <parameterUnits>ft</parameterUnits>
            </Parameter>
        </otherTestProperty>
    </diggs_geo:StaticConePenetrationTest>
</procedure>
```

This is added as an “otherTestProperty.” We can have a variable number of these, and they add a lot of flexibility to the “diggs_geo”StaticConePenetrationTest” object. It houses a “parameter” object to which we add three fields “parameterName,” “parameterValue,” and “parameterUnits.” The “parameterName” is for defining what this property is, so we set it to “Assumed Water Depth.” The value is recorded in “parameterValue” and we add the unit of measure to the “parameterUnits” property.

But what about recording finding the water depth? It is similar, but we use a “testEvent” property with an “InSituTestEvent” child:

```
<procedure>
    <diggs_geo:StaticConePenetrationTest gml:id="cpt_DGS88D1-BC6-3E5F-7E94-
68907">
        <otherTestProperty>
            <Parameter gml:id="wd_DGS1AE6-17D-3911-A256-272FA">
                <parameterName>Assumed Water Depth</parameterName>
                <parameterValue>6.00</parameterValue>
                <parameterUnits>ft</parameterUnits>
            </Parameter>
        </otherTestProperty>
        <testEvent>
            <InSituTestEvent gml:id="wd1_DGSC01-189F-15F5-913A-2B952">
                <infoRecorded>
                    <Parameter gml:id="wd2_DGS3D0E-78-3B99-ECD8-63FBC">
                        <parameterName>Water Depth</parameterName>
                        <parameterValue>6.00</parameterValue>
                        <parameterUnits>ft</parameterUnits>
                    </Parameter>
                </infoRecorded>
            </InSituTestEvent>
        </testEvent>
    </diggs_geo:StaticConePenetrationTest>
</procedure>
```

To the “InsituTestEvent” we just add one property “infoRecorded” which contains a “Parameter” object that works in the same way as above. With that, we are in the home stretch. Let’s look at the last handful of properties for recording how the CPT testing was done.

```xml
<procedure>
    <diggs_geo:StaticConePenetrationTest gml:id="cpt_DGS88D1-BC6-3E5F-7E94-
68907">
        <testEvent>
            <InSituTestEvent gml:id="wd1_DGSC01-189F-15F5-913A-2B952">
                <infoRecorded>
                    <Parameter gml:id="wd2_DGS3D0E-78-3B99-ECD8-63FBC">
                        <parameterName>Water Depth</parameterName>
                        <parameterValue>6.00</parameterValue>
                        <parameterUnits>ft</parameterUnits>
                    </Parameter>
                </infoRecorded>
            </InSituTestEvent>
        </testEvent>
        <diggs_geo:penetrometerType>piezocone</diggs_geo:penetrometerType>
        <diggs_geo:distanceTipToSleeve
uom="cm">15</diggs_geo:distanceTipToSleeve>
        <diggs_geo:netAreaRatioCorrection>0.8</diggs_geo:netAreaRatioCorrection>
        <diggs_geo:penetrationRate uom="cm/s">1</diggs_geo:penetrationRate>
        <diggs_geo:tipArea uom="cm2">15</diggs_geo:tipArea>
    </diggs_geo:StaticConePenetrationTest>
</procedure>
```

Firstly, we record the type of the penetrometer with the “diggs_geo:penetreometerType” property. Then the distance from the tip to sleeve is recorded with the “diggs_geo:distanceTipToSleeve” and the area ratio correction in the “diggs_geo:netAreaRatioCorrection” property. Finally, the penetration rate and tip area are recorded with the “diggs_geo:penetrationRate” and the “diggs_geo:tipArea” properties respectively. Each of these last two need a “uom” attribute to define the units of measure being used.

Atterberg Limits Test

Recording data for an Atterberg Limits Test follows the same fundamental concepts seen previously, and it just varies in its expression of the geometry of recorded data and the types of data recorded. Let’s take a look:

```xml
<measurement>
    <Test gml:id="Atterberg_SS-1">
        <gml:name>Atterberg Limits, SS-1</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
        <samplingFeatureRef xlink:href="#Location_B-001-0-20" />
        <outcome>
            <TestResult gml:id="Result_Atterberg_SS-1">
                <location>
                    <LinearExtent
                        srsDimension="1"
                        srsName="B-001-0-20-lsr"
                        gml:id="DGS14C8-DDA-156-D66F-1E2F3">
                        <gml:posList>1.50 2.58</gml:posList>
                    </LinearExtent>
                </location>
                <results>
                    <!-- Abridged XML -->
                </results>
            </TestResult>
        </outcome>
    </Test>
</measurement>
```

The fields used for identifying the Test such as “gml:name,” “investigationTarget,” “projecetRef” and “samplingFeatureRef” work the same as previously. Similarly, a “LinearExtent” object is used to define the portion of the borehole relevant for the test in the same way as above. The differences come into play when we zoom in on the “results” property as below:

```
<results>
    <ResultSet>
        <parameters>
            <PropertyParameters gml:id="DGS134F-8EC-4B2B-F8A2-25D58">
                <properties>
                    <Property index="1" gml:id="DGS9775-10B9-4CB3-4E31-4484A">
                        <propertyName>LL</propertyName>
                        <typeData>integer</typeData>
                        <propertyClass
                            codeSpace="https://www.diggsml.org/dictionaries/DIGGS
TestPropertyDefinitions.xml">
                            liquid_limit</propertyClass>
                    </Property>
                    <Property index="2" gml:id="DGS92DE-31C-2EC1-C5DD-5C7F9">
                        <propertyName>PL</propertyName>
                        <typeData>integer</typeData>
                        <propertyClass
                            codeSpace="https://www.diggsml.org/dictionaries/DIGGS
TestPropertyDefinitions.xml">
                            plastic_limit</propertyClass>
                    </Property>
                    <Property index="3" gml:id="DGS216F-7DF-4F43-59E2-4AD61">
                        <propertyName>PI</propertyName>
                        <typeData>integer</typeData>
                        <propertyClass
                            codeSpace="https://www.diggsml.org/dictionaries/DIGGS
TestPropertyDefinitions.xml">
                            plasticity_index</propertyClass>
                    </Property>
                </properties>
            </PropertyParameters>
        </parameters>
        <dataValues cs="," decimal="." ts=" ">24, 20,4</dataValues>
    </ResultSet>
</results>
```

The structure of the results section is the same as for our previous examples, but the types of measurements that we're making differ. In this example, we have XML to record the liquid limit, plastic limit, and plasticity index. The data itself is recorded as a series of comma separated values.

Finally, let's look at the procedure section for an Atterberg Limits Test:

```xml
<measurement>
    <Test gml:id="Atterberg_SS-1">
        <procedure>
            <diggs_geo:AtterbergLimitsTest gml:id="DGS64C5-11BF-1903-3296-6864E">
                <testProcedureMethod>
                    <Specification gml:id="AASHTO_Y-89">
                        <gml:name>AASHTO T 89, 2022 Edition, Standard Method of
Test for Determining the Liquid Limit of Soils</gml:name>
                        <accredtingBody>AASHTO</accredtingBody>
                    </Specification>
                </testProcedureMethod>
                <testProcedureMethod>
                    <Specification gml:id="AASHTO_T-90">
                        <gml:name>AASHTO T 90, Standard Method of Test for Deter-
mining the Plastic Limit and Plasticity Index of Soils</gml:name>
                        <accredtingBody>AASHTO</accredtingBody>
                    </Specification>
                </testProcedureMethod>
            </diggs_geo:AtterbergLimitsTest>
        </procedure>
    </Test>
</measurement>
```

Firstly, we have “testProcedureMethod” properties with “Specification” children. These work in the same way as we saw for our SPT testing, but they correspond to the methods used for the measurements we took here. After this, there are a couple of values unique to our “diggs_geo:AtterbergLimitsTest” object that we need to employ. Let’s look at them now:

```xml
<measurement>
    <Test gml:id="Atterberg_SS-1">
        <procedure>
            <diggs_geo:AtterbergLimitsTest gml:id="DGS64C5-11BF-1903-3296-6864E">
                <specimen>
                    <SoilSpecimen gml:id="Spec_Sample_B-001-0-20_1.50_1_SS_">
                        <sampleRef xlink:href="#Sample_B-001-0-20_1.50_1_SS_" />
                    </SoilSpecimen>
                </specimen>
                <diggs_geo:percentRetainedNo40>30</diggs_geo:percentRetainedNo40>
            </diggs_geo:AtterbergLimitsTest>
        </procedure>
    </Test>
</measurement>
```

Firstly, the “specimen” property has a “SoilSpecimen” child. This is used to connect the Atterberg test to its corresponding sample which is connected through the “sampleRef” property using an “xlink:href” attribute. The value of this attribute is the “gml:id” of the sample preceded by a “#” character.

Finally, we use the “diggs_geo:percentRetainedNo40” property to set a retention of 30%.

Conclusion

And with that, we are done! We have recorded SPT, CPT, and Atterberg Limit tests in DIGGS. If you are following along with these guides in order, next we will be looking at how to use DIGGS XML to record Water Level Measurement Results for an Installed Piezometer. See you there!

Water Level Measurement Results for Installed Piezometer

Introduction

In this guide, we will be looking at how to document a well installed inside of a borehole. Afterwards, we'll be looking at recording water level measurements from an installed piezometer. The example XML used in this guide are loosely based on some mock results from The Ohio Department of Transportation (ODOT). It is based on data obtained from a borehole log that can be found here. The actual test site this borehole log did not require installing a well in a borehole, so liberties were taken for illustrative purposes.

Documenting a Well

Wells in DIGGS are a type of installation, they are installed inside of a sampling feature which in our case is a borehole. To begin, we will get some basic identifying information written up. Let's look at some XML:

```
<samplingFeature>
    <Well gml:id="Piezometer_B-001-0-20">
        <gml:description>Standpipe Piezometer installed in B-001-0-
20</gml:description>
        <gml:name>B-001-0-20 Piezometer</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
    </Well>
</samplingFeature>
```

The "samplingFeature" above is a property of the root "Diggs" element. Our well uses the "Well" object type as its only child. To this, we add a "gml:id," "gml:description," and a "gml:name." These are commonly recurring elements that you can find out more about in the Using DIGGS Identifiers article found here.

The "investigationTarget" property uses a string to indicate the target of the investigation, but the list of possible values is restricted to one of the following:

- Deep Foundation - investigation of deep foundation elements
- Earthworks -earth materials excavated and/or emplaced by engineering activity
- Ground Water -hydrology and chemistry of ground water;
- Indoor Atmosphere - meteorology and chemistry of air in indoor spaces
- Natural Ground - earth materials sampled in natural or undisturbed state;
- Outdoor Atmosphere - meteorology and chemistry of air in outdoor spaces
- Surface Water - hydrology and chemistry of surface water;

Then finally, we connect this well to its associated project with the "projectRef" property. The "xlink:href" attribute connects to the project by referencing its "gml:id" property preceded by a, "#" character.

Next up, we need to define the dimensionality of the well. We will do this with a few different objects. The first will be used to define the location of where the top of the well is accessed. Let's look at an example:

```
<samplingFeature>
    <Well gml:id="Piezometer_B-001-0-20">
        <gml:description>Standpipe Piezometer installed in B-001-0-20</gml:de-
scription>
        <gml:name>B-001-0-20 Piezometer</gml:name>
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
        <referencePoint>
            <PointLocation gml:id="p2-B-001-0-20">
                <gml:pos
                    srsDimension="3"
                    srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
                    uomLabels="dega dega ftUS"
                    axisLabels="latitude longitude height">
                    39.474660 -81.796858 819.6
                </gml:pos>
            </PointLocation>
        </referencePoint>
    </Well>
</samplingFeature>
```

Next, the "referencePoint" property's purpose is to note the position of, in this case, the well. We do that with a property from GML "gml:pos." A lot of work is being done here with the attributes; "srsDimensions" is set to, "3" as we will be using latitude, longitude, and height. This format for our special reference system is referenced with our special reference system (srs) name attribute "srsName." The above value denotes we are using WGS284 latitude and longitude for the first two dimensions then NGVD29 height in feet for the third.

The "uomLabels" attribute has a series of unit of measure labels one for each dimension listed in order. In this case "dega" appears twice for our degree measurements, then "height" is used for the height. Similarly "axisLabels" labels what the three dimensions are to avoid ambiguity. Finally, the actual latitude, longitude, and elevation are written in order as the value of the "gml:pos" object, separated by spaces.

You may have noticed that the well is a bit higher than the borehole it is installed in. That is because the standpipe is located above the ground surface.

Next, we will look at using a center line to define the geometry of the well:

```xml
<samplingFeature>
    <Well gml:id="Piezometer_B-001-0-20">
        <centerLine>
            <LinearExtent gml:id="clp-B-001-0-20">
                <gml:posList
                    srsDimension="3"
                    srsName="urn:diggs:def:crs:DIGGS:0.1:4326_5702"
                    uomLabels="dega dega ftUS"
                    axisLabels="latitude longitude height">
                    39.474660 -81.796858 819.6 39.474660 -81.796858 799.6
                </gml:posList>
            </LinearExtent>
        </centerLine>
    </Well>
</samplingFeature>
```

Now we have a new “centerLine” property with a “LinearExtent” object to define the geometry of our well. It’s similar to the previous “referencePoint” property, but there are some key differences. Our “LinearExtent” object has a “gml:posList” property which represents a list of positions. We need to use one for the top of the well and one for the bottom. It has the same attributes with the same values as our “gml:pos” object from the previous example because we’re using the same coordinate system for mapping the borehole. The value of the “gml:posList” is also a space separated list, but the first three values are for the top of the borehole, and the second set of three values are for the bottom. Because the structure of the data is defined, there is not a need to use a new line for each row of data as might be expected in a traditional CSV file.

```xml
<Well gml:id="Piezometer_B-001-0-20">
    <linearReferencing>
        <LinearSpatialReferenceSystem gml:id="lsrp-B-001-0-20">
            <gml:identifier
                codeSpace="urn:x-def:authority:DIGGSINC">
                urn:x-diggs:def:fi:DIGGSINC:B-001-0-20-lsrp
            </gml:identifier>
            <glr:linearElement xlink:href="#cl-B-001-0-20"/>
            <glr:lrm>
                <glr:LinearReferencingMethod gml:id="lrmp-B-001-0-20">
                    <glr:name>chainage</glr:name>
                    <glr:type>absolute</glr:type>
                    <glr:units>ft</glr:units>
                </glr:LinearReferencingMethod>
            </glr:lrm>
        </LinearSpatialReferenceSystem>
    </linearReferencing>
</Well>
```

In the preceding code, the “LinearSpatialReferenceSystem” can be seen. It is used within our well to contextualize the information of our “LinearExtent” object in our “centerLine.” The “gml:identifier” is required for this object, and we use it to denote that DIGGS is defining this information. Next, we use the “glr:linearElement” property to connect this object back to its “LinearExtent” using the “xlink:href” attribute. As before, whenever using an href we append a “#” prefix to the gml:id of the target object.

We define the linear referencing method inside the “glr:lrm” property with a “glr:LinearReferencingMethod” object. It has three properties “glr:name,” “glr:type,” and “glr:units,” which we fill with chainage, absolute, and feet (“ft”) as is appropriate to our use case.

Finally, we will connect to Well to the relevant borehole and define aspects of its construction. Let’s get into it:

```
<Well gml:id="Piezometer_B-001-0-20">
    <samplingFeature>
        <samplingFeatureRef xlink:href="#Location_B-001-0-20" />
        <opening>
            <WellOpening gml:id="wo1">
                <openInterval>
                    <LinearExtent gml:id="wo1l" srsDimension="1" srsName="#lsrp-
B-001-0-20">
                        <gml:posList>19 20</gml:posList>
                    </LinearExtent>
                </openInterval>
            </WellOpening>
        </opening>
        <wellCasing>
            <Casing gml:id="wc1">
                <casingInsideDiameter uom="in">2</casingInsideDiameter>
                <casingMaterial>PVC</casingMaterial>
            </Casing>
        </wellCasing>
        <wellDepth uom="ft">20</wellDepth>
        <wellPurpose>water level monitoring</wellPurpose>
    </samplingFeature>
</Well>
```

The “samplingFeatureRef” property works just like the property reference from before, but in this case we use it to connect this well to the borehole that it is installed in. We define where its opening is using the “opening” property with a “WellOpening” object. We need one property “openInterval” with a “LinearExtent” object. This is the same type of object as before, but in this we only need one dimension. The “gml:posList” defines points along the well’s center line.

The “wellcasing” property with its “Casing” object defines the well casing’s diameter and material with the “casingInsideDiameter” object and the “casingMaterial” properties respectively. For the diameter, using the “uom” property to define the unit of measure is important, and in this case is set to inches. This allows the actual value to be a numeric data type more readily parsable when it is later extracted.

With that, we now have a basic well installed inside of a borehole! If you would like to see the above code totally assembled into one sampling feature without any sections abridged, you can find that XML here. Next up, we will be recording some water level measurements! See you below.

Recording Water Level Measurement Results

To begin, let’s look at some of the identifying information for our “Monitor” object:

```xml
<measurement>
    <Monitor gml:id="Water_Levels_Piezometer_B-001-0-20">
        <investigationTarget>Natural Ground</investigationTarget>
        <projectRef xlink:href="#Project_600819" />
        <sampleRef xlink:href="#Piezometer_B-001-0-20" />
        <reading>
            <!-- Abridged XML -->
        </reading>
    </Monitor>
</measurement>
```

The “investigationTarget” and the “projectRef” properties were in the same manner as above in the preceding section, but it is worth noting that the “sampleRef” might be considered a bit unintuitive. In this case, the “sample” we are referencing is actually the well with our installed piezometer. We use the “gml:id” we established for the well above to connect these monitor results to it.

The rest of the example code is going to take place entirely inside of the “reading” property. This “Reading” object contained inside requires quite a bit of horizontal space, so we will be zoomed all the way into this section to preserve space.

First, we will look at establishing when measurements took place:

```xml
<Reading gml:id="Reading_Water_Levels_Piezometer_B-001-0-20">
    <responseZoneLocationRef xlink:href="#wo1l" />
    <outcome>
        <MonitorResult gml:id="Result_Water_Levels_Piezometer_B-001-0-20">
            <timeDomain>
                <TimePositionList gml:id="TP_Water_Levels_Piezometer_B-001-0-20">
                    <timePositionList>
                        2021/01/30
                        2021/02/14
                        2021/03/01
                        2021/03/15
                        2021/03/20
                    </timePositionList>
                </TimePositionList>
            </timeDomain>
        </MonitorResult>
    </outcome>
</Reading>
```

Most of the data for this “Reading” object is contained inside the “outcome” property’s “MonitorResult” object. The “timeDomain” is similar in some ways to a Linear Extent in that it has a series of values defining an extent of something. In this case though those points are in time. For each recording that took place, the date is recorded in YYYY/MM/DD format within the “timePositionList.”

With this, we have a series of points of time in which we know that a reading occurred, but there is no actual data. That is not very helpful, so we will remedy that by adding a “ResultSet” as below:

```
<Reading gml:id="Reading_Water_Levels_Piezometer_B-001-0-20">
    <responseZoneLocationRef xlink:href="#wo1l" />
    <outcome>
        <MonitorResult gml:id="Result_Water_Levels_Piezometer_B-001-0-20">
            <results>
                <ResultSet>
                    <parameters>
                        <PropertyParameters gml:id="pp_Water_Levels_Piezometer_B-
001-0-20">
                            <properties>
                                <Property index="1"
gml:id="prop1_Water_Levels_Piezometer_B-001-0-20">
                                    <gml:name>Depth to Water</gml:name>
                                    <typeData>double</typeData>
                                    <propertyClass
                                        codeSpace="http://diggsml.org/dictionarie
s/DIGGSTestPropertyDefinitions.xml">
                                        depth_to_water</propertyClass>
                                    <uom>ft</uom>
                                </Property>
                            </properties>
                        </PropertyParameters>
                    </parameters>
                    <dataValues>
                        8.2 8.5 7.9 8.0 8.1
                    </dataValues>
                </ResultSet>
            </results>
        </MonitorResult>
    </outcome>
</Reading>
```

Firstly, the “paremeters” property with its “PropertyParameters” child is used to define what kind of data we are recording later in our result set. In this case, we only need one “Property” for our “properties” collection because there is only one type of data being recorded here. How deep is it to get to water? We define that in human readable terms using the “gml:name” property. The “typeData” property defines the type used for recording the data. Here, it is set to “double.” This references a type of numeric data known as a float the explanation of which is outside the scope of this guide.

The “propertyClass” parameter references a dictionary with a controlled list of properties that are used to define more precisely in a fixed manner a number of possible properties with defined qualities. The dictionary defined above is a real one that you can view online here. The “gml:Id” of the relevant definition is used as the value of this property to specify which value in the dictionary is being used.

We finish defining the results set by setting the unit of measure with the “uom” property. The “ft” value means the results are in feet.

Finally, we record the actual data! Use the “dataValues” property to include a series of values separated by spaces for each level water was recorded at. These values correspond, in order, to the series of time values we defined previously. They match up one to one.

With that, we are almost done. We just have one more thing we need to document, our sensor:

```xml
<Reading gml:id="Reading_Water_Levels_Piezometer_B-001-0-20">
    <responseZoneLocationRef xlink:href="#wo1l" />
    <outcome>
        <!-- Abridged XML -->
    </outcome>
    <sensor>
        <Sensor gml:id="tape">
            <gml:name>steel tape</gml:name>
            <class>tape</class>
        </Sensor>
    </sensor>
</Reading>
```

Below our “outcome” property is a “sensor” property with a “Sensor” object. We just need two of its properties: “gml:name” which denotes it is a steel tape sensor, and a “class” property which defines that this is a “tape” sensor. Ideally, your organization should create a controlled list of which possible values can appear in this “class” value. It is intended to define the type of equipment being used.

If at some point you felt confused about how these highlighted code samples fit together, you can find a complete unabridged example of the XML from this section here.

Conclusion

With that, we are finished! If you have been following along, you have now document a well installed of a borehole and recorded water level results taken over a period of time. If you are following along with this guide in order, the final entry is up next where we will cover transitioning from getting started to exploring the full DIGGS schema using the official documentation. See you there!

Next Steps

Introduction

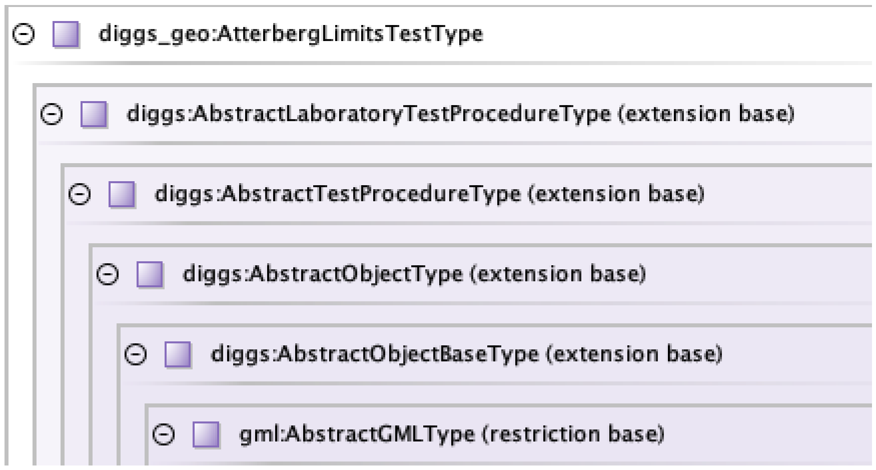
In this final section of the guide, we will cover some fundamentals of how DIGGS work that have not been address previously as well as how to use the full DIGGS documentation to create DIGGS XML not covered by this getting started guide.

This getting started guide is based on the 2.6 release of the DIGGS XML Standard, but the fundamental concepts will remain true into future versions. The full documentation for the latest public release of DIGGS can be found at https://diggsml.github.io/. As you follow along with this guide, you should have this page open, and you're encouraged to peruse the documentation on your own. We will cover

Abstract Types and Object Inheritance

The DIGGS schema is incredibly flexible because it needs to account for all varieties of geotechnical data that organizations store and transmit. Part of how that is possible is thanks to a concept called, "Object Inheritance." This is a concept likely familiar to you if you've written in an object-oriented language before. Essentially, the idea is that one type can, "inherit" from another type. Whenever one type inherits from another, it gains all of the properties of its parent type. For example, the "gml:AbstractGMLType" found in the documentation has four child properties three of which are "gml:description," "gml:identifier," and "gml:name." These should look familiar as they've reoccurred in many places throughout our DIGGS schema. That is because DIGGS objects inherit from this "gml:AbstractGMLType." They do not do this directly though.

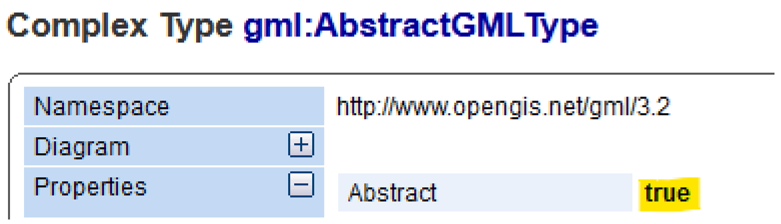
A child may inherit from other multiple types by inheriting in a chain one level at a time. Let's look at an example:



```
diggs_geo:AtterbergLimitsTestType
  diggs:AbstractLaboratoryTestProcedureType (extension base)
    diggs:AbstractTestProcedureType (extension base)
      diggs:AbstractObjectType (extension base)
        diggs:AbstractObjectBaseType (extension base)
          gml:AbstractGMLType (restriction base)
```

Here, the "diggs_geo:AtterbergLimitsTest" inherits from "diggs:AbstractLaboratoryTestProcedureType" which inherits from "diggs:AbstractTextProcedureType" and so on all the way one at a time until we reach the familiar "gml:AbstractGMLType" that we were looking at before. So, even though there are four layers of inheritance between "diggs_geo:AtterbergLimitsTestType" and "gml:AbstractGMLType" we still have "gml:name" as a property for our Atterberg test. It is worth noting that at no point does one level contain two different types. Each type only directly inherits one other type. So, that being said, why are there so many different types that all start with, "Abstract?"

There are a number of types in the documentation that you cannot directly use which are known as abstract types. Any type which begins (after the namespace prefix) with, "Abstract" is an abstract type that is not available for use within a DIGGS document. You can tell that these are abstract because they have a property of "Abstract" that is set to true. Let's look at an example from the schema for the "gml:AbstractGMLType."


Complex Type gml:AbstractGMLType

Namespace	http://www.opengis.net/gml/3.2	
Diagram ⊞		
Properties ⊟	Abstract	true

Here, you can see that "Abstract" is set to, "true."

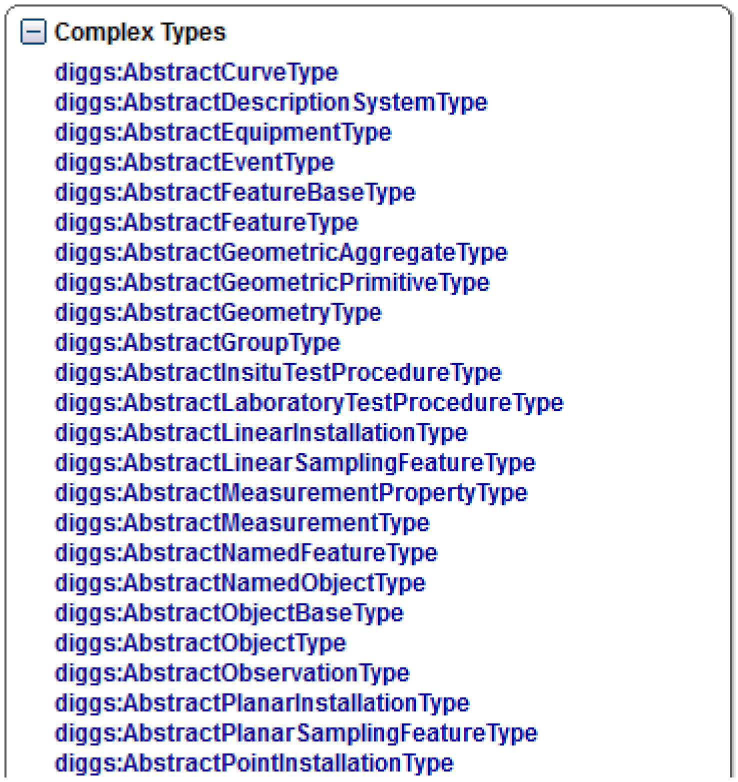
So, DIGGS has a number of abstract types, but why does that matter? Essentially, these abstract types are used as a foundation for other types to build upon. They are a useful way of storing a related set of properties, but they are not in themselves a complete concept, and instead they need to be expanded upon. Any object which has a complex type property that calls for an abstract type as a child can be relied upon to have certain features regardless of which implementation of that abstract type is used. Later, after the DIGGS XML has been created and transmitted, these common fields can be very easily parsed with a minimal implementation.

Substitution Groups

Substitution groups are used to fulfil a similar purpose of increasing flexibility while maintaining limits on the added complexity. Substitution groups are a collection of types that can each be used in a particular place, but do not necessarily have to share certain attributes in common. This allows for a known structure, but it is more flexible because if a future update to the schema alters the structure of one member of a substitution group, the others are unaffected. You can tell which substitution groups, if any, a DIGGS object belongs to by referencing its "Substitution Group Affiliation" schema property. If we return to our "diggs_geo:AtterbergLimitsTest" example, we can see it is a member of the "diggs:AbstractLaboratoryTestProcedure" substitution group. This interchangeability powers the procedure property that we first encountered in the Recording SPT, CPT, and Atterberg Limit Testing in DIGGS guide previously.

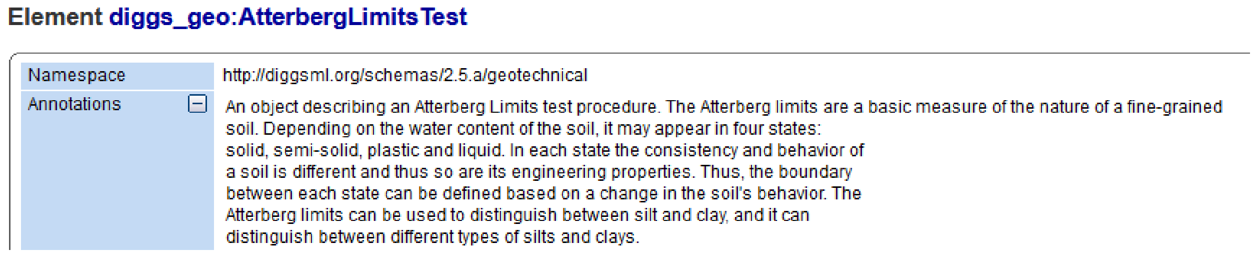
Navigating the DIGGS Documentation

Let's return to the index. The DIGGS documentation contains the complete breakdown of all the types and elements in the XML schema. These are broken down, by default, into a number of lists the most relevant of which are "Elements," "Complex Types," and "Simple Types." You can find an explanation of the difference between complex and simple types along with the object-property rule, here.


Complex Types

diggs:AbstractCurveType
diggs:AbstractDescriptionSystemType
diggs:AbstractEquipmentType
diggs:AbstractEventType
diggs:AbstractFeatureBaseType
diggs:AbstractFeatureType
diggs:AbstractGeometricAggregateType
diggs:AbstractGeometricPrimitiveType
diggs:AbstractGeometryType
diggs:AbstractGroupType
diggs:AbstractInsituTestProcedureType
diggs:AbstractLaboratoryTestProcedureType
diggs:AbstractLinearInstallationType
diggs:AbstractLinearSamplingFeatureType
diggs:AbstractMeasurementPropertyType
diggs:AbstractMeasurementType
diggs:AbstractNamedFeatureType
diggs:AbstractNamedObjectType
diggs:AbstractObjectBaseType
diggs:AbstractObjectType
diggs:AbstractObservationType
diggs:AbstractPlanarInstallationType
diggs:AbstractPlanarSamplingFeatureType
diggs:AbstractPointInstallationType

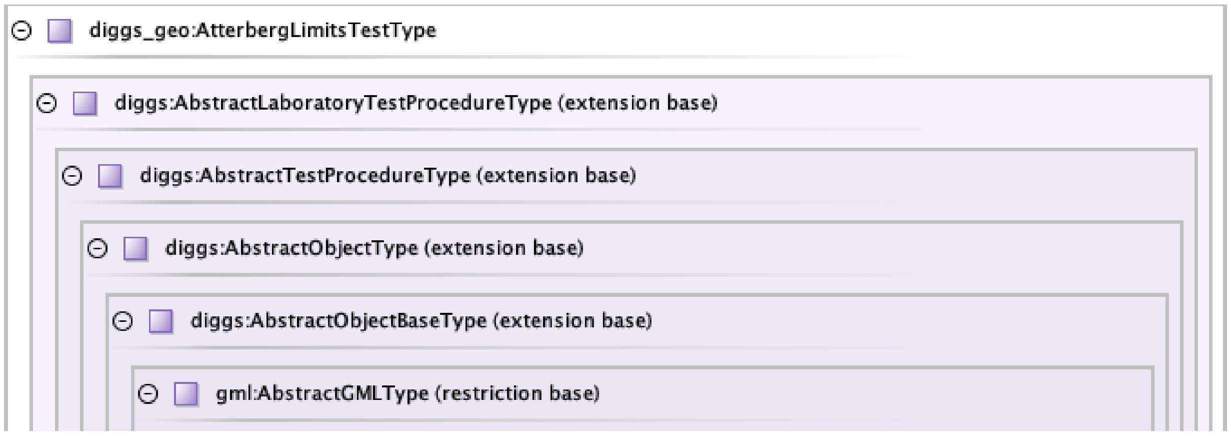
To get more information on any of these, just click on the appropriate value in the table of contents. For the purposes of this example, we will be clicking the, diggs_geo:AtterbergLimitsTest complex type object from before. Which will bring us to the record for this complex type. It looks something like this:


Element diggs_geo:AtterbergLimitsTest

Namespace	http://diggsml.org/schemas/2.5.a/geotechnical
Annotations	An object describing an Atterberg Limits test procedure. The Atterberg limits are a basic measure of the nature of a fine-grained soil. Depending on the water content of the soil, it may appear in four states: solid, semi-solid, plastic and liquid. In each state the consistency and behavior of a soil is different and thus so are its engineering properties. Thus, the boundary between each state can be defined based on a change in the soil's behavior. The Atterberg limits can be used to distinguish between silt and clay, and it can distinguish between different types of silts and clays.

The namespace disambiguates the meaning of the XML namespace prefix in front of our complex type. These example images were captured from the 2.5 version of the documentation, so you may notice an older namespace reference.

The annotations property is used as a description for what this object does. What is arguably the most useful section of the documentation is the next one, the diagram of the object. Let's take a look first with the familiar:



diggs_geo:AtterbergLimitsTestType

diggs:AbstractLaboratoryTestProcedureType (extension base)

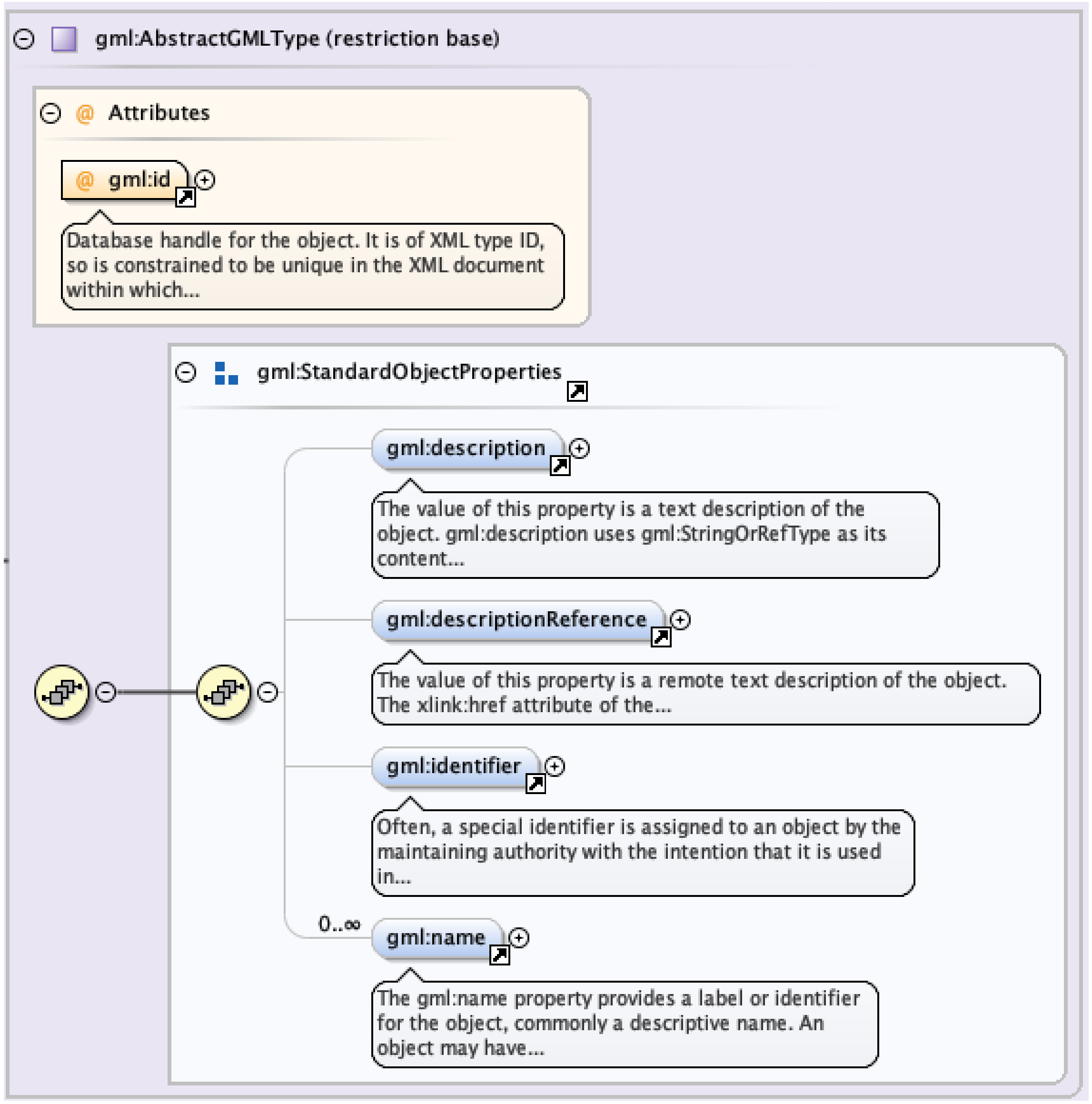
diggs:AbstractTestProcedureType (extension base)

diggs:AbstractObjectType (extension base)

diggs:AbstractObjectBaseType (extension base)

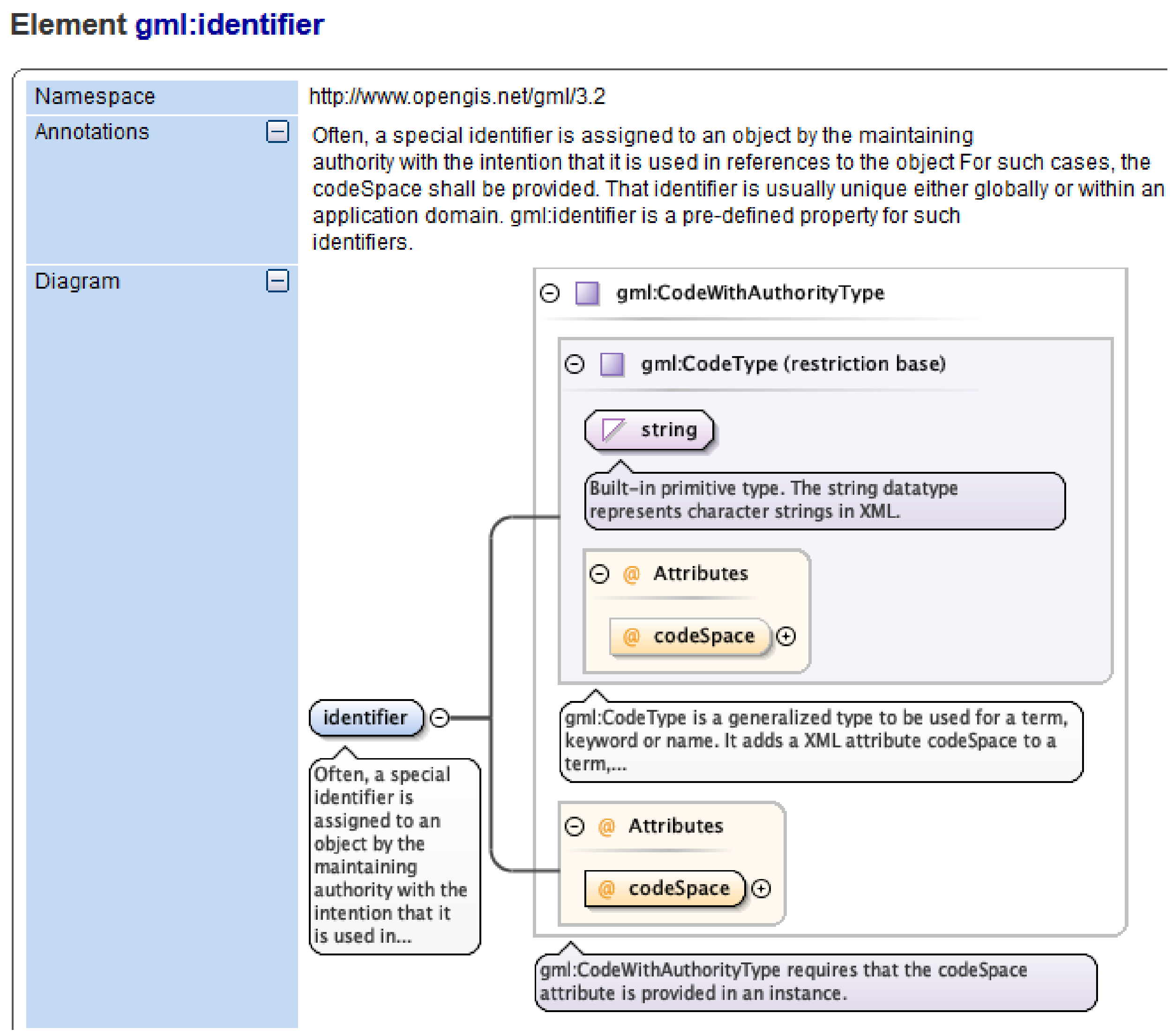
gml:AbstractGMLType (restriction base)

We have a nested series of types reflected in the inheritance our object has. Each child contains the parent nested within itself, so as we drill deeper into the diagram, we are higher in the hierarchy. Each box contains the attributes and properties for that type. We will start with the innermost type:


gml:AbstractGMLType (restriction base)
Attributes
gml:id
Database handle for the object. It is of XML type ID, so is constrained to be unique in the XML document within which...
gml:StandardObjectProperties
gml:description
The value of this property is a text description of the object. gml:description uses gml:StringOrRefType as its content...
gml:descriptionReference
The value of this property is a remote text description of the object. The xlink:href attribute of the...
gml:identifier
Often, a special identifier is assigned to an object by the maintaining authority with the intention that it is used in...
0..∞
gml:name
The gml:name property provides a label or identifier for the object, commonly a descriptive name. An object may have...

As you can see, the attributes appear first, as in this case with the “gml:id” attribute, then properties follow in a separate box. The symbol next to the “gml:name” property indicates that any number of “gml:name” properties between zero and theoretically infinity of the “gml:name” property may appear within the object. This isn’t specified for the other three properties, so the same is not true for them; instead, these may have either one or zero of each of these values. If one of them were required to have exactly one, the line would be bolded. If at least one were required, the zero in the zero to infinity symbol would instead be a one.

Any of these properties can be clicked to learn more about what they are, or what values they require. If we click “gml:identifier” for example, it will take us here:

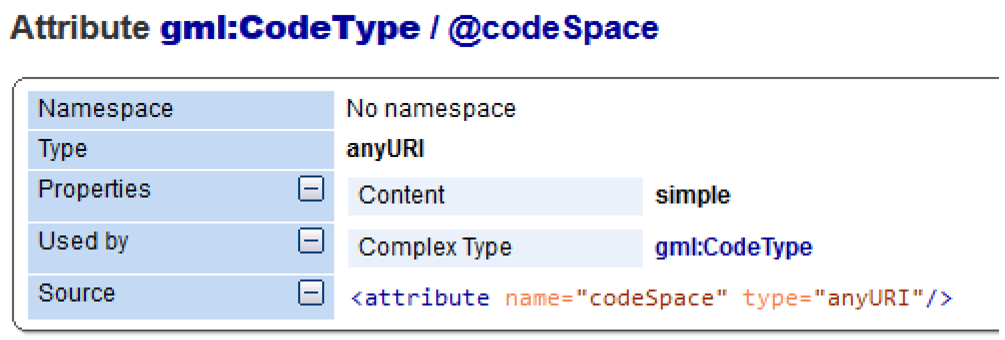


Element gml:identifier

Namespace	http://www.opengis.net/gml/3.2
Annotations	Often, a special identifier is assigned to an object by the maintaining authority with the intention that it is used in references to the object For such cases, the codeSpace shall be provided. That identifier is usually unique either globally or within an application domain. gml:identifier is a pre-defined property for such identifiers.
Diagram	

gml:CodeWithAuthorityType
gml:CodeType (restriction base)
string
Built-in primitive type. The string datatype represents character strings in XML.
Attributes
codeSpace
gml:CodeType is a generalized type to be used for a term, keyword or name. It adds a XML attribute codeSpace to a term,...
identifier
Often, a special identifier is assigned to an object by the maintaining authority with the intention that it is used in...
Attributes
codeSpace
gml:CodeWithAuthorityType requires that the codeSpace attribute is provided in an instance.

By viewing the diagram for this property, we can see its child is a string, and it has a “codeSpace” attribute. The bold line pointing to the string value and the codeSpace attribute indicates that the DIGGS schema requires both of these when a “gml:identifier” is present. We can drill in even further and look at the codeSpace attribute if we want as below.

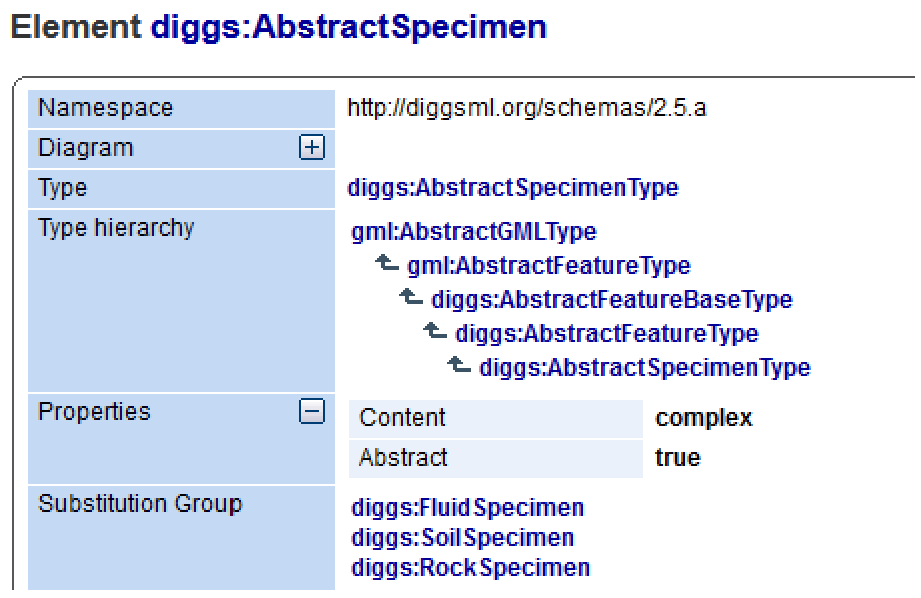


Attribute gml:CodeType / @codeSpace

Namespace	No namespace
Type	**anyURI**
Properties	Content: **simple**
Used by	Complex Type: gml:CodeType
Source	`<attribute name="codeSpace" type="anyURI"/>`

This attribute is fairly straightforward, so there is less value in having done so, but it serves to illustrate the navigability of the schema.

Returning to our Atterberg Limits Test object from before, there is a “specimen” property of that is inherited from the “diggs:AbstractLaboratoryTestProcedureType” type. Let’s assume we want to figure out what to place there. We will drill into it by clicking “specimen” which brings us to here. Its child is a “diggs:AbstractSpecimen” type. Well, this might cause a problem if we remember from before that we cannot directly use an abstract type in our DIGGS schema, but fortunately we have a solution. Click “diggs:AbstractSpecimen” to arrive at its record. Here, we have an easy way to see all the implementations of this via the “Substitution Groups” schema property seen here:



Element diggs:AbstractSpecimen

Namespace	http://diggsml.org/schemas/2.5.a
Diagram	
Type	diggs:AbstractSpecimenType
Type hierarchy	gml:AbstractGMLType ⮤ gml:AbstractFeatureType ⮤ diggs:AbstractFeatureBaseType ⮤ diggs:AbstractFeatureType ⮤ diggs:AbstractSpecimenType
Properties	Content: **complex** Abstract: **true**
Substitution Group	diggs:FluidSpecimen diggs:SoilSpecimen diggs:RockSpecimen

So, we can use either a “diggs:FluidSPecimen,” “diggs:SoilSpecimen,” or “diggs:RockSpecimen.” This technique of drilling into the schema helps to isolate one piece of information at a time that we need when building DIGGS XML documents.

Conclusion

With that, you have all the tools you need to continue exploring DIGGS and creating XML documents to house your various kinds of geotechnical data. It can be helpful to review prior examples of the XML created in previous sections of this guide, and to review the full XML documentation for each piece to better understand how things fit together, but from here you're ready to explore and experiment with the full schema.