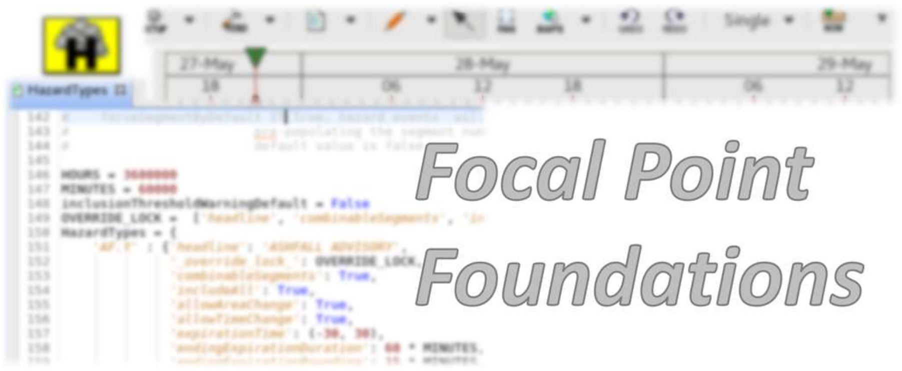
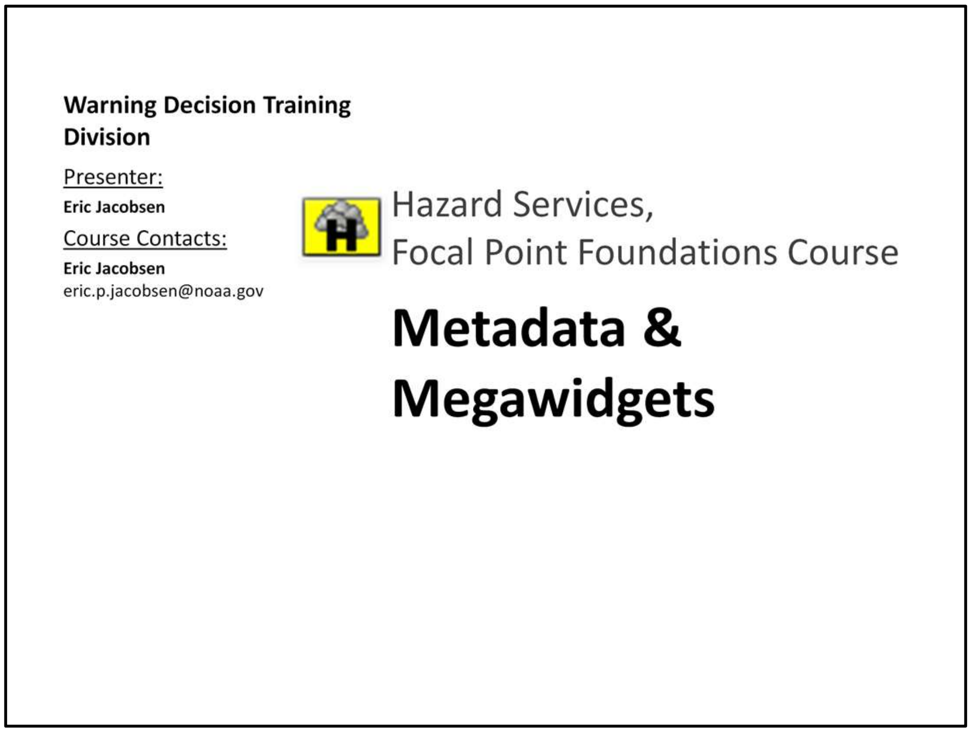


Focal Point Foundations

Click "Next" to begin the presentation…

[No audio for this slide]



Warning Decision Training Division

Presenter:

Eric Jacobsen

Course Contacts:

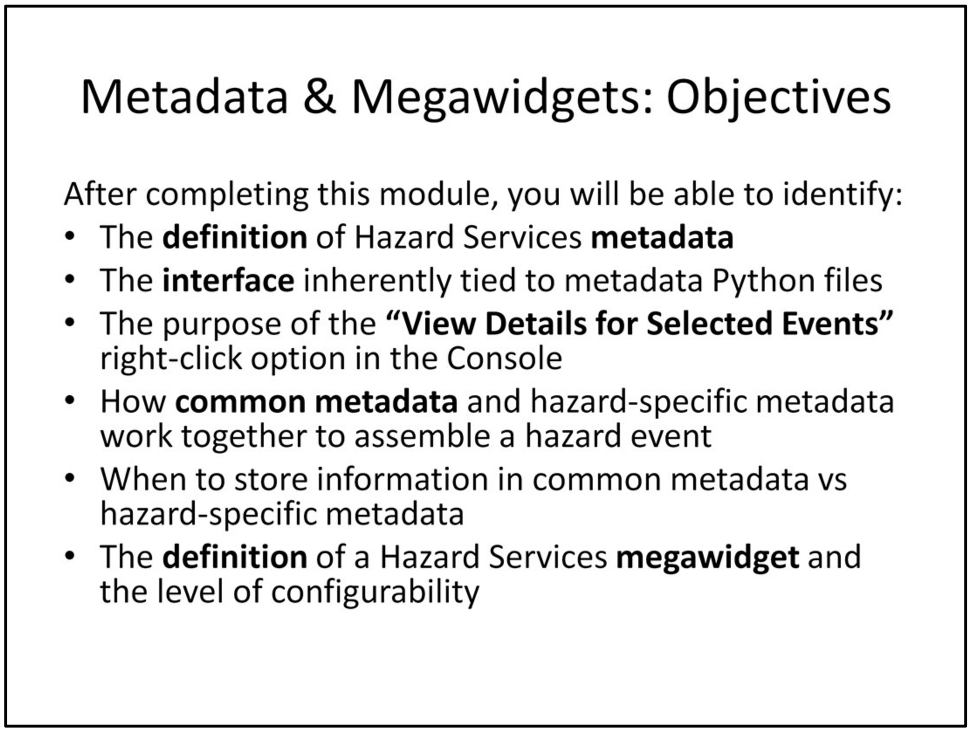
Eric Jacobsen
eric.p.jacobsen@noaa.gov

Hazard Services,
Focal Point Foundations Course

Metadata & Megawidgets

Welcome to the module covering Metadata and its closely related topic, Megawidgets, part of the Hazard Services Foundational training course for Focal Points.

My name is Eric Jacobsen, with the Warning Decision Training Division. If you have questions about this course, or technical problems, please use the contact information listed on this slide.

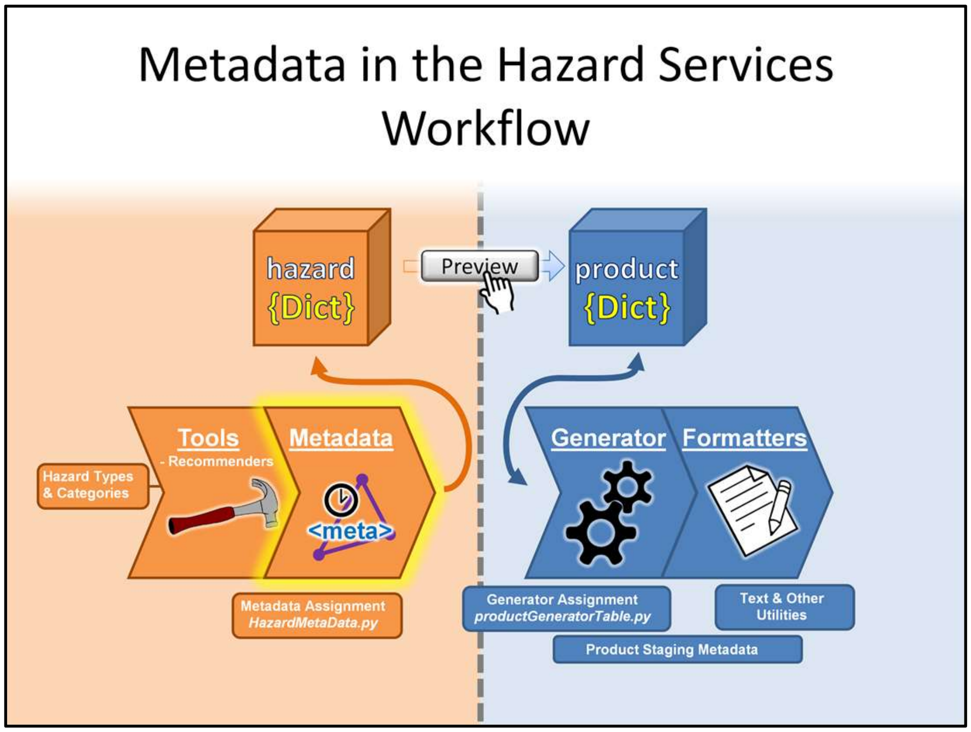


Metadata & Megawidgets: Objectives

After completing this module, you will be able to identify:

- The **definition** of Hazard Services **metadata**
- The **interface** inherently tied to metadata Python files
- The purpose of the **“View Details for Selected Events”** right-click option in the Console
- How **common metadata** and hazard-specific metadata work together to assemble a hazard event
- When to store information in common metadata vs hazard-specific metadata
- The **definition** of a Hazard Services **megawidget** and the level of configurability

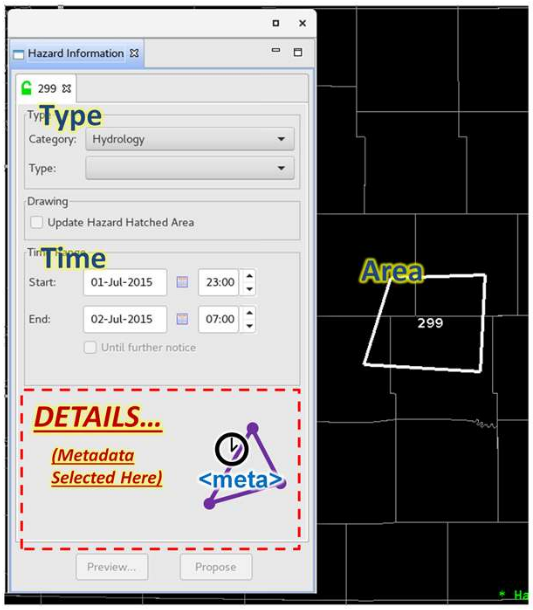
These are the objectives for this module. Please take a moment to review them, then, when you’re done, click next to proceed with the module.



Metadata in the Hazard Services Workflow

Metadata are one of several major configurable "components" in the Hazard Services workflow. Here is a reminder of the overall Hazard Services workflow, initially presented in the introductory, unified workflow module, with Metadata highlighted here

As we'll soon explore in greater detail, recall that, in the hazard vs product division which underlies Hazard Services, metadata are principally relevant to characterizing Hazard events.



Category: Hydrology

Type:

Update Hazard Hatched Area

Start: 01-Jul-2015 23:00

End: 02-Jul-2015 07:00

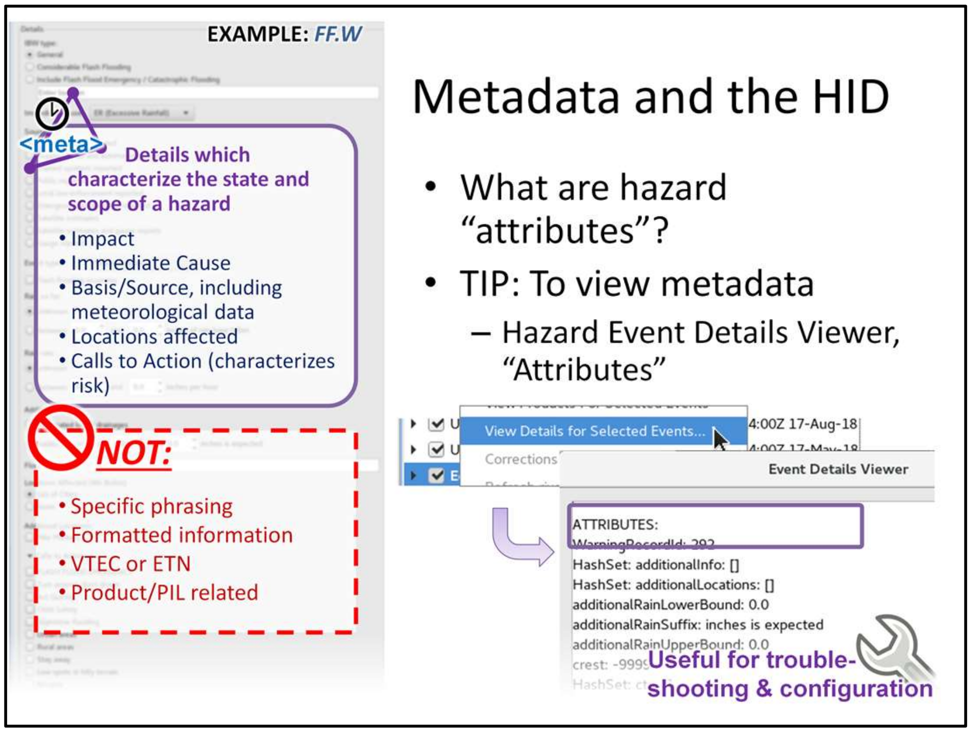
Until further notice

What is Metadata?

- “Atomic” attributes of hazard
  - Key component of Hazard Event Dictionary
- Hazard Information Dialogue
  - “Details” section
  - Empty if metadata not provided

Hazard Services defines metadata as the “atomic” attributes of a hazard. We’ll get into what that means in more detail in the next slides, but recall that a fully-defined hazard requires: a hazard type, an area (or geographic information), a time rang, and, finally, other details to characterize it.

Except for area, these qualities are closely tied to the Hazard Information Dialogue, and in fact the “details” section which makes up the bulk of the HID’s usefulness, is exactly the interface for specifying a hazard’s metadata.



EXAMPLE: *FF.W*

Metadata and the HID

- What are hazard "attributes"?
- TIP: To view metadata
  - Hazard Event Details Viewer, "Attributes"

<meta> Details which characterize the state and scope of a hazard

- Impact
- Immediate Cause
- Basis/Source, including meteorological data
- Locations affected
- Calls to Action (characterizes risk)

NOT:

- Specific phrasing
- Formatted information
- VTEC or ETN
- Product/PIL related

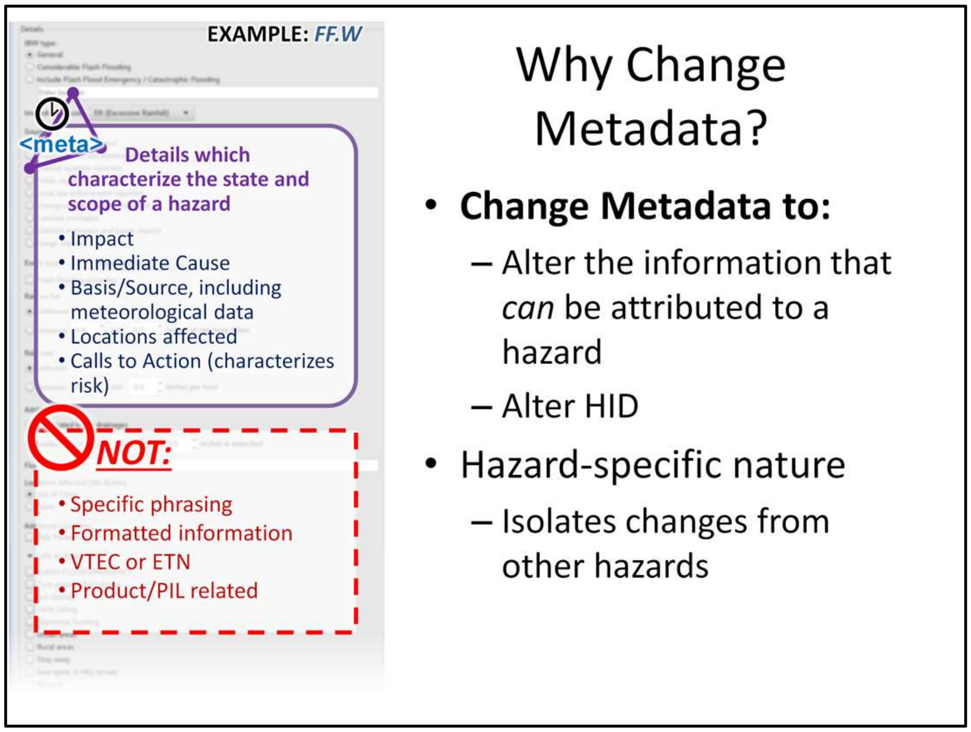
```
ATTRIBUTES:
WarningRecordId: 292
HashSet: additionalInfo: []
HashSet: additionalLocations: []
additionalRainLowerBound: 0.0
additionalRainSuffix: inches is expected
additionalRainUpperBound: 0.0
crest: -9999
```

Useful for trouble-shooting & configuration

Here is an example of the "details" section of the HID for a Flash Flood Warning, where we see the types of attributes that can make up hazard metadata. Again, the selections made on the Hazard Information Dialogue directly translate to the metadata for the hazard.

To summarize the common nature of attributes which can be selected here, metadata are details which characterize the state and scope of a hazard, including but not limited to: impacts, immediate cause, basis or source, other descriptive data fields, and even calls-to-action if they characterize the risk specific to that hazard. As "atomic" hazard information, metadata is not concerned with specific phrasing, formatting, or even the type of product or VTEC information which are later used to communicate about it.

To understand more about hazard metadata, users may right-click on a hazard in their console and select "View Details for Selected Events". The Event Details Viewer which appears will, under "ATTRIBUTES," show all the metadata fields attributed to that hazard. This supplementary view at the hazard event is most likely to be of interest when troubleshooting hazards, or when going "under-the-hood" to learn about configuration opportunities.



EXAMPLE: *FF.W*

<meta>

Details which characterize the state and scope of a hazard

- Impact
- Immediate Cause
- Basis/Source, including meteorological data
- Locations affected
- Calls to Action (characterizes risk)

NOT:

- Specific phrasing
- Formatted information
- VTEC or ETN
- Product/PIL related

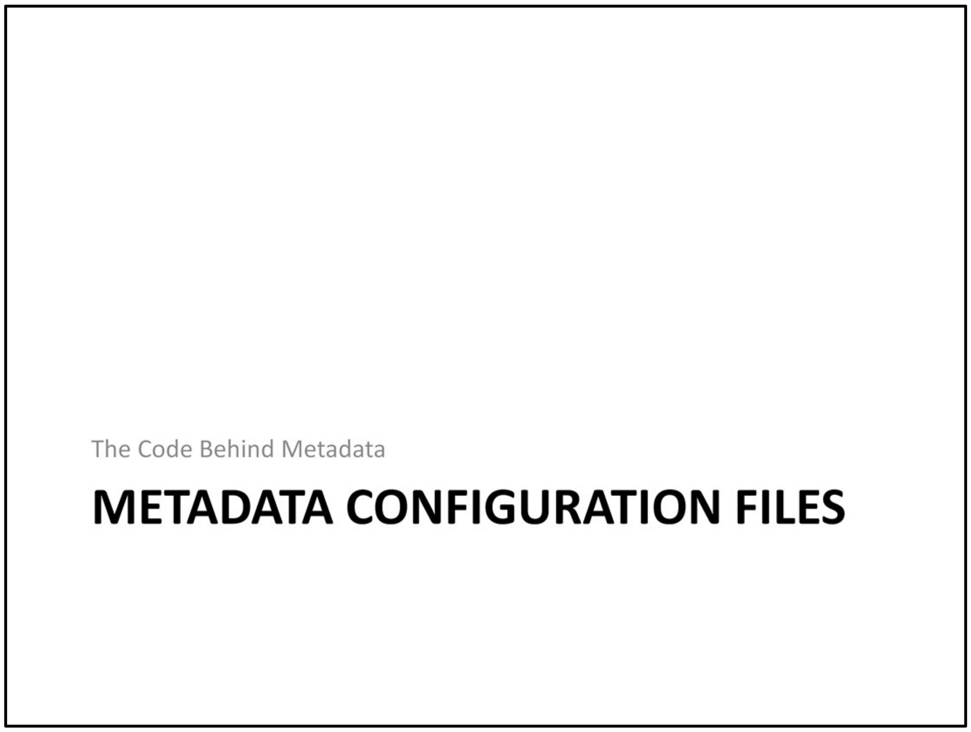
Why Change Metadata?

- **Change Metadata to:**
  - Alter the information that *can* be attributed to a hazard
  - Alter HID
- Hazard-specific nature
  - Isolates changes from other hazards

So how does “metadata” matter to focal points?

The purpose of configuring metadata is to change information that can be attributed to a hazard, which can include adding or removing attributes, changing choices, or even changes to the behavior of fields or widgets used to specify these attributes. Effectively, you change the metadata to change the Hazard Information Dialogue for a hazard.

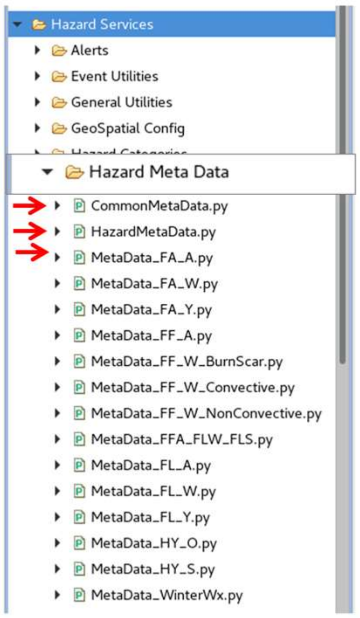
It’s important that Metadata are almost always hazard-specific. That is, most metadata files are tailored to a one or at most a small handful of hazards with nearly identical attribute needs. That affords focal points the flexibility to edit the HID for a specific hazard, to implement custom behavior, without affecting other hazards.



The Code Behind Metadata

METADATA CONFIGURATION FILES

Let’s talk about how metadata configuration files actually work.



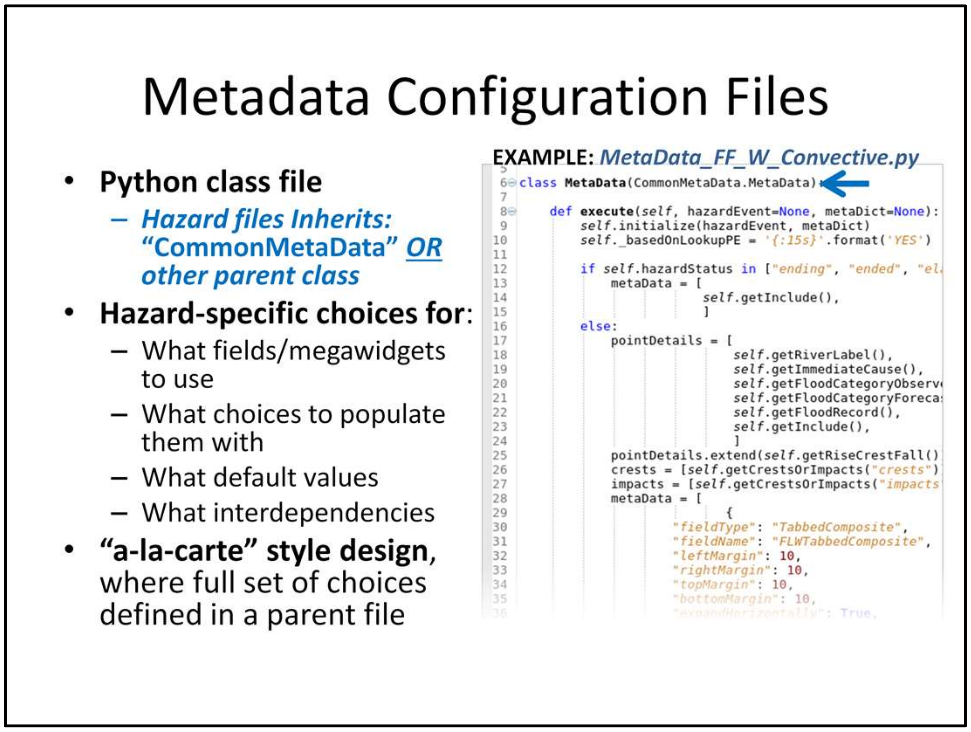
Where to Find Metadata

- Localization Perspective: Hazard Meta Data
- Contents:
  - Hazard-specific files
  - Shared files
- Verify which Metadata file maps to a hazard through:
  - HazardMetaData.py

Metadata configurations are accessible through the localization perspective under the “Hazard Metadata” subdirectory of “Hazard Services”.

Overall, multiple files here represent the specification of metadata for each hazard type. In addition, a few of the files seen here are “common” or shared among multiple hazards. In particular, we point out “CommonMetaData.py” which is a foundation on which many of the hazard-specific metadata are built, as we’ll see shortly.

Although the file titles are usually descriptive enough, a focal point can ensure that they’re editing the appropriate metadata file for their hazard by reviewing the “HazardMetadata.py” file and finding the precise file mapping for their hazard type. This relatively straightforward configuration file typically dedicates a line for mapping each metadata file to the one or more hazard types which it should apply to. Crafted with the nested dictionary structure we’re familiar with from the Python overview, it is therefore easily modifiable with incremental overrides.



Metadata Configuration Files

- **Python class file**
  - ***Hazard files Inherits: "CommonMetaData" OR other parent class***
- **Hazard-specific choices for**:
  - What fields/megawidgets to use
  - What choices to populate them with
  - What default values
  - What interdependencies
- **"a-la-carte" style design**, where full set of choices defined in a parent file

EXAMPLE: *MetaData_FF_W_Convective.py*

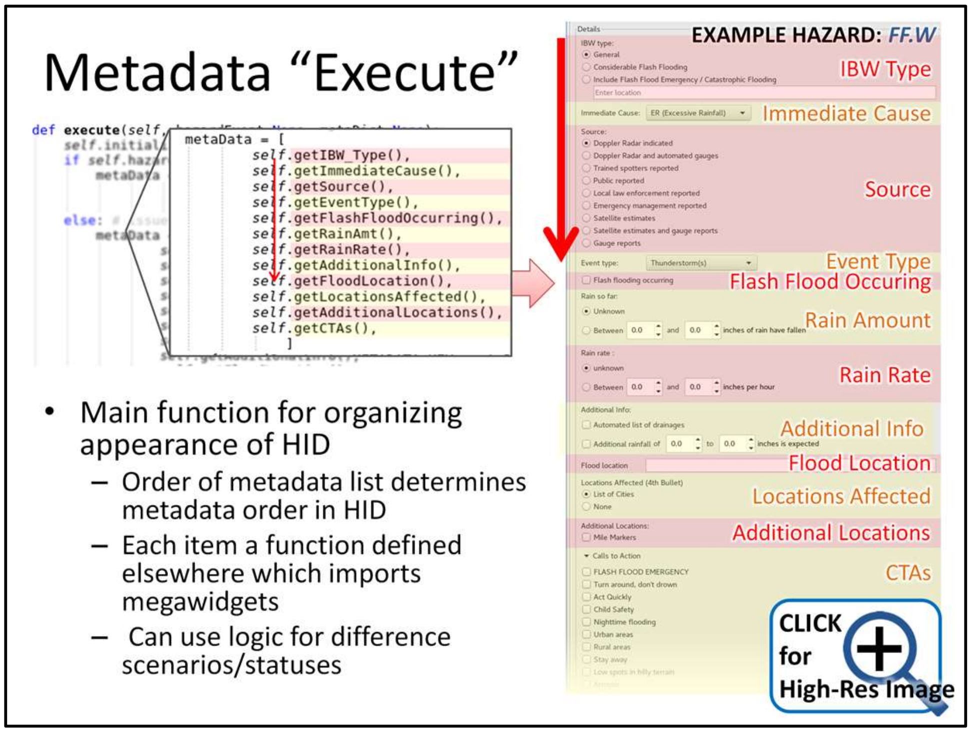
```
class MetaData(CommonMetaData.MetaData):

    def execute(self, hazardEvent=None, metaDict=None):
        self.initialize(hazardEvent, metaDict)
        self._basedOnLookupPE = '{:15s}'.format('YES')

        if self.hazardStatus in ["ending", "ended", "el
            metaData = [
                            self.getInclude(),
                            ]
        else:
            pointDetails = [
                                self.getRiverLabel(),
                                self.getImmediateCause(),
                                self.getFloodCategoryObserv
                                self.getFloodCategoryForeca
                                self.getFloodRecord(),
                                self.getInclude(),
                                ]
            pointDetails.extend(self.getRiseCrestFall()
            crests = [self.getCrestsOrImpacts("crests")
            impacts = [self.getCrestsOrImpacts("impacts
            metaData = [
                                {
                        "fieldType": "TabbedComposite",
                        "fieldName": "FLWTabbedComposite",
                        "leftMargin": 10,
                        "rightMargin": 10,
                        "topMargin": 10,
                        "bottomMargin": 10,
                        "expandHorizontally": True,
```

Each metadata file is a python file which, primarily, follows a class definition model. This class definition, recalling the python overview module, relies on a parent template, which is inherited by including it in parentheses on the first line. In this case, we see that CommonMetaData is the template or "parent class" used as a starting point for this file, which means that all the capabilities of CommonMetaData are inherited without any further effort.

Most of each hazard-specific metadata file concerns itself with specifying the exact fields, choices, values, and behavior needed for its given hazard. We'll explain later how many hazard-specific metadata files are built with an "a-la-carte" style design, where they simply select from the options made available to them by their parent metadata file.



Metadata "Execute"

```
def execute(self,
    self.initial
    if self.hazar
        metaData

    else: #
        metaData
```

```
metaData = [
            self.getIBW_Type(),
            self.getImmediateCause(),
            self.getSource(),
            self.getEventType(),
            self.getFlashFloodOccurring(),
            self.getRainAmt(),
            self.getRainRate(),
            self.getAdditionalInfo(),
            self.getFloodLocation(),
            self.getLocationsAffected(),
            self.getAdditionalLocations(),
            self.getCTAs(),
                ]
```

- Main function for organizing appearance of HID
  - Order of metadata list determines metadata order in HID
  - Each item a function defined elsewhere which imports megawidgets
  - Can use logic for difference scenarios/statuses

EXAMPLE HAZARD: *FF.W*

Details

IBW type: [✓ General] Considerable Flash Flooding / Include Flash Flood Emergency / Catastrophic Flooding
Enter location

Immediate Cause: ER (Excessive Rainfall)

Source: [✓ Doppler Radar indicated] Doppler Radar and automated gauges / Trained spotters reported / Public reported / Local law enforcement reported / Emergency management reported / Satellite estimates / Satellite estimates and gauge reports / Gauge reports

Event type: Thunderstorm(s)

☐ Flash flooding occurring

Rain so far: [✓ Unknown] Between 0.0 and 0.0 inches of rain have fallen

Rain rate : [✓ unknown] Between 0.0 and 0.0 inches per hour

Additional Info:
☐ Automated list of drainages
☐ Additional rainfall of 0.0 to 0.0 inches is expected

Flood location

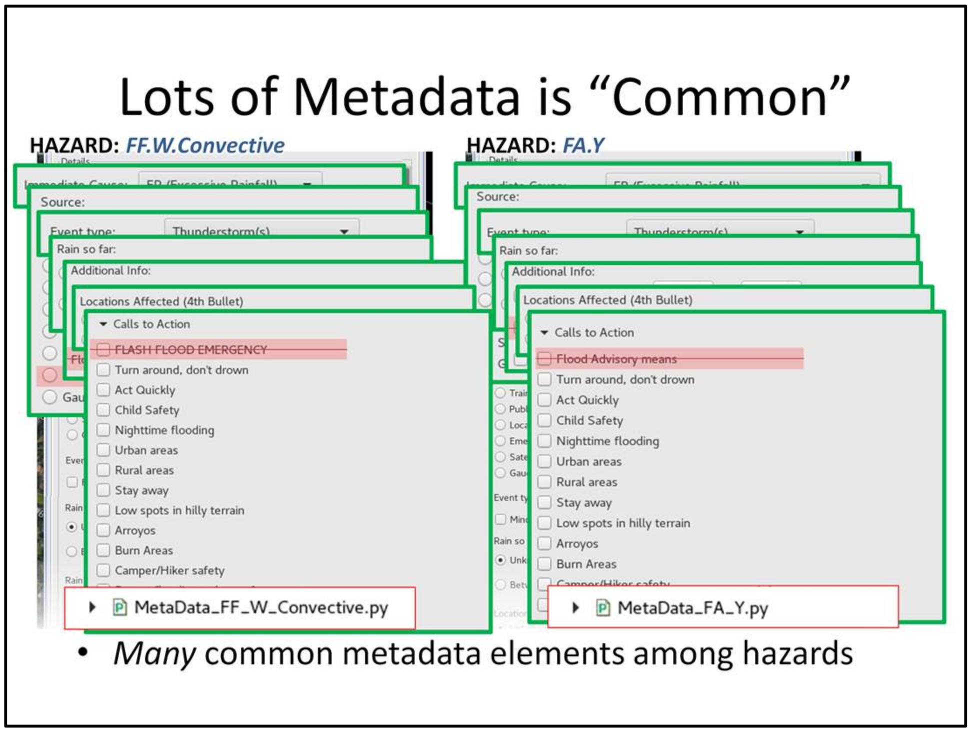
Locations Affected (4th Bullet): [✓ List of Cities] None

Additional Locations:
☐ Mile Markers

Calls to Action
☐ FLASH FLOOD EMERGENCY
☐ Turn around, don't drown
☐ Act Quickly
☐ Child Safety
☐ Nighttime flooding
☐ Urban areas
☐ Rural areas
☐ Stay away
☐ Low spots in hilly terrain

The previously stated relation of the metadata file to the HID is especially clear if we look at the main "execute" method which every metadata file must include.

This execute method contains, most notably, the crucial calls to assemble the HID "details" section, one-by-one, though functions that generate the modular interfaces known as megawidgets. You can see that each sub-interface of the HID matches perfectly to the similarly-named functions calls, including their order.



Lots of Metadata is "Common"

HAZARD: *FF.W.Convective*

HAZARD: *FA.Y*

- *Many* common metadata elements among hazards

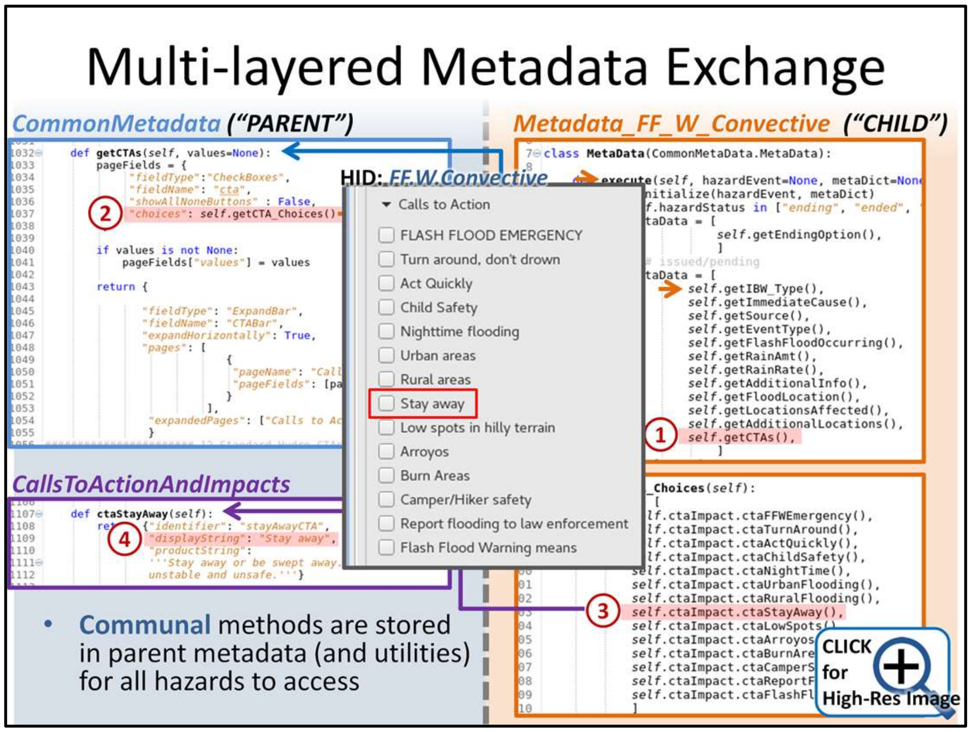
There is a crucial pattern to how metadata files are organized which deserves emphasis.

In general, each hazard has unique requirements for the layout and detailed options within the HID. These two side-by-side HIDS for two different hazards do have differences, as expected. Put another way, each hazard benefits from having its own, targeted metadata definition file, and as previously stated, that's typically the case.

However, this understates the similar functionality that many hazards, especially in a common category, like Hydrology, can have. For example, as is illustrated by the generally identical elements being highlighted on the screen, many hazards use similar megawidgets, draw from identical call-to-action choices or other phrases, and use the same back-end methods.

Although not all hazards will have as much in common as these two do, it's still often the case that, while the overall assembly of metadata might differ from hazard to hazard, they draw from a common pool of "building blocks", and it's senseless to define the exact same methods in every hazard file.

Hazard Services therefore adopts a multi-tiered metadata scheme, with common functionality stored in a higher-level file that hazard-specific metadata can reference as needed. In fact, this layered approach is not just unique to metadata… a tiered organization for configuration is used in many other components of Hazard Services, such as the Product Generation workflow, but that’s covered in a separate section.



Multi-layered Metadata Exchange

CommonMetadata *("PARENT")*

```
    def getCTAs(self, values=None):
        pageFields = {
                "fieldType":"CheckBoxes",
                "fieldName": "cta",
                "showAllNoneButtons" : False,
                "choices": self.getCTA_Choices()

        if values is not None:
            pageFields["values"] = values

        return {

                  "fieldType": "ExpandBar",
                  "fieldName": "CTABar",
                  "expandHorizontally": True,
                  "pages": [
                          {
                           "pageName": "Call
                           "pageFields": [pa
                          }
                      ],
                   "expandedPages": ["Calls to Ac
                   }
```

CallsToActionAndImpacts

```
    def ctaStayAway(self):
        ret    {"identifier": "stayAwayCTA",
                "displayString": "Stay away",
                "productString":
                '''Stay away or be swept away.
                unstable and unsafe.'''}
```

HID: *FF.W.Convective*

Metadata_FF_W_Convective *("CHILD")*

```
class MetaData(CommonMetaData.MetaData):

        execute(self, hazardEvent=None, metaDict=None
        nitialize(hazardEvent, metaDict)
        f.hazardStatus in ["ending", "ended",
        taData = [
                    self.getEndingOption(),
                    ]
        # issued/pending
        taData = [
                self.getIBW_Type(),
                self.getImmediateCause(),
                self.getSource(),
                self.getEventType(),
                self.getFlashFloodOccurring(),
                self.getRainAmt(),
                self.getRainRate(),
                self.getAdditionalInfo(),
                self.getFloodLocation(),
                self.getLocationsAffected(),
                self.getAdditionalLocations(),
                self.getCTAs(),
                    ]
```

```
        _Choices(self):
         [
        lf.ctaImpact.ctaFFWEmergency(),
        lf.ctaImpact.ctaTurnAround(),
        lf.ctaImpact.ctaActQuickly(),
        lf.ctaImpact.ctaChildSafety(),
       self.ctaImpact.ctaNightTime(),
       self.ctaImpact.ctaUrbanFlooding(),
       self.ctaImpact.ctaRuralFlooding(),
       self.ctaImpact.ctaStayAway(),
       self.ctaImpact.ctaLowSpots()
       self.ctaImpact.ctaArroyos
       self.ctaImpact.ctaBurnAre
       self.ctaImpact.ctaCamperS
       self.ctaImpact.ctaReportF
       self.ctaImpact.ctaFlashFl
       ]
```

- **Communal** methods are stored in parent metadata (and utilities) for all hazards to access

What does multi-layered metadata look like in practice?

Let's analyze where how one very common megawidget, that is, the sub-interface on the HID, for the calls to action is produced, using the flash flood hazard type for illustration. In doing so, we'll initially bounce between two files. On the right is the hazard-specific metadata file for the convective FFW type. On the left is "commonMetadata," which is a large collection of shared resources and method definitions used by many hazards, as we'll shortly see. In fact, let's specially reserve the left side of the screen for centralized files, as we'll need to involve another one shortly, versus files which are tailored to a single hazard, on the right.

We have already reviewed that the execute function is central to assembling each hazard's custom HID, so it's no surprise that the hazard metadata file dictates interface components, which includes (last in the list, and last on the HID) the "getCTAs" request, highlighted here.

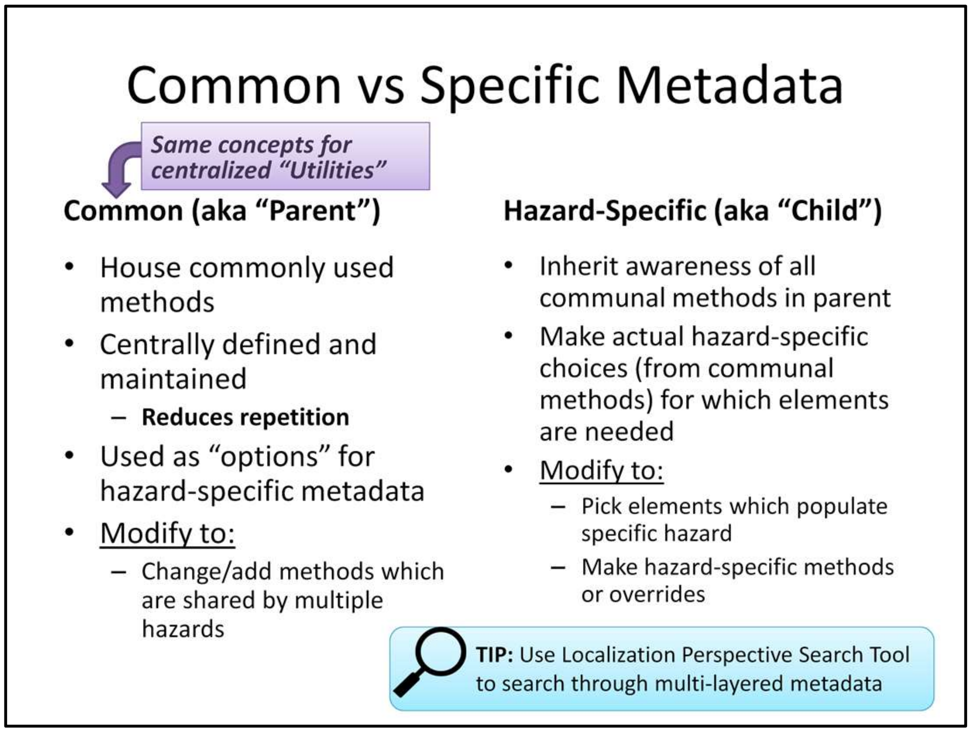
Now, "getCTAs" provides the general definition of the megawidget for calls to action. We've seen that almost every hazard uses this megawidget, so it's only practical to place this definition in a central place, accessible by all. Thus it finds a home in the common metadata, rather than being duplicated in every derivative hazard file.

Next, the CTA interface can't be assembled without choices for the checkboxes, hence an embedded function call to getCTAChoices. Because the CTA choices should ultimately match the hazard, it should make sense that the hazard metadata file this time defines this function, with a list of choices that suits its needs. Sure enough, we can see this function simply returning a list of more functions, each of which in particular appears to be related to populating a specific call-to-action

Picking the “ctaStayAway” method to follow one step further, we find we have to look in a different file which holds all cta and impact statements, but a shared one very much like common metadata. This single, centralized file is naturally called “CallsToActionAndImpacts.py”, and resides in the “utilities” directory. Let’s disregard for now HOW this file is linked, and follow the thread to our desired CTA…

Now we’re finally at the end of the ride… in the CallsToActionAndImpacts file, we discover that the ctaStayAway function returns a dictionary consisting of several text strings, one of which is, at last, the label used to populate “Stay Away “ in the Call to Action megawidget. Success!

But why was the last link in this chain, the call-to-action definition itself, stored in a separate, centralized file and not in the hazard-specific one itself? Once again, though it might be used by the FFW hazard type, “stay away” is a frequently used call-to-action displayed by many hazards. Defining this and most of the CTAs in a central file is a matter of practicality, since there they are defined and maintained in only one place, where they can be accessed by many hazards.



Common vs Specific Metadata

Same concepts for centralized "Utilities"

Common (aka "Parent")

- House commonly used methods
- Centrally defined and maintained
  - **Reduces repetition**
- Used as "options" for hazard-specific metadata
- Modify to:
  - Change/add methods which are shared by multiple hazards

Hazard-Specific (aka "Child")

- Inherit awareness of all communal methods in parent
- Make actual hazard-specific choices (from communal methods) for which elements are needed
- Modify to:
  - Pick elements which populate specific hazard
  - Make hazard-specific methods or overrides

TIP: Use Localization Perspective Search Tool to search through multi-layered metadata

Stepping back, and summarizing the important points…

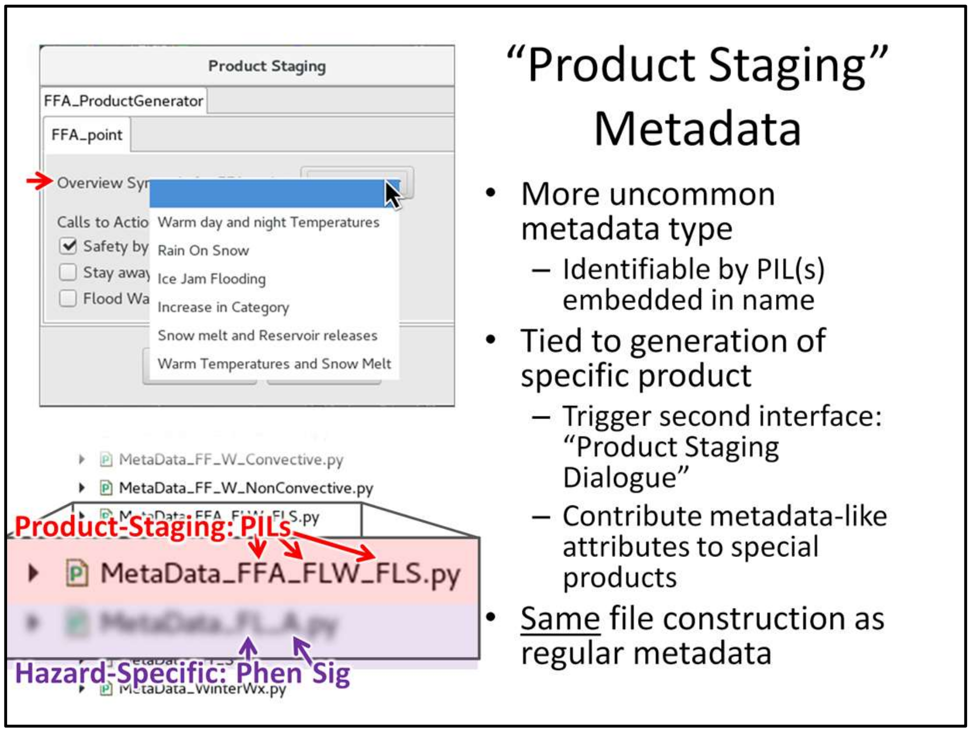
Multi-layered metadata exists because many aspects of metadata serve a common purpose and are shared across hazards. A parent metadata class housing any method which is more communal than hazard-specific is a convenient, single point of maintenance. Many megawidget definitions, choices, and other functions are housed in these parent metadata classes.

In turn, hazard-specific metadata, which are python classes, inherit this common class by design when they are created, gaining full access to the communal methods defined within. This, by the way, is a GREAT real-world example of the implementation and advantages of the class inheritance topic covered in the python overview, which might understandably have seemed a little abstract when first introduced.

Now, the utilities file for calls to action and impacts is a little bit of an oddball, since it doesn't strictly fit the "metadata class" mold. But what's more important than the mechanism of how it's linked, is that, Although the CallsToActionAndImpacts file may come from a non-metadata directory, like CommonMetadata, it represents a centralized side of Metadata Management, which cooperates with hazard-specific files for multi-layered metadata construction.

This structure provides order and prevents redundancy in the metadata file set, but it may cause problems for Focal Points trying to trace a function to its roots. To that end, keep the localization perspective search tool handy, since it can be used to easily find function references across files.

Finally, when making configuration changes in a multi-layered metadata framework, Focal Points should strive to differentiate between changes that are truly hazard-specific and those which are likely to be multi-purposed, and if the latter, use the common or parent metadata classes to house those capabilities.



"Product Staging" Metadata

- More uncommon metadata type
  - Identifiable by PIL(s) embedded in name
- Tied to generation of specific product
  - Trigger second interface: "Product Staging Dialogue"
  - Contribute metadata-like attributes to special products
- Same file construction as regular metadata

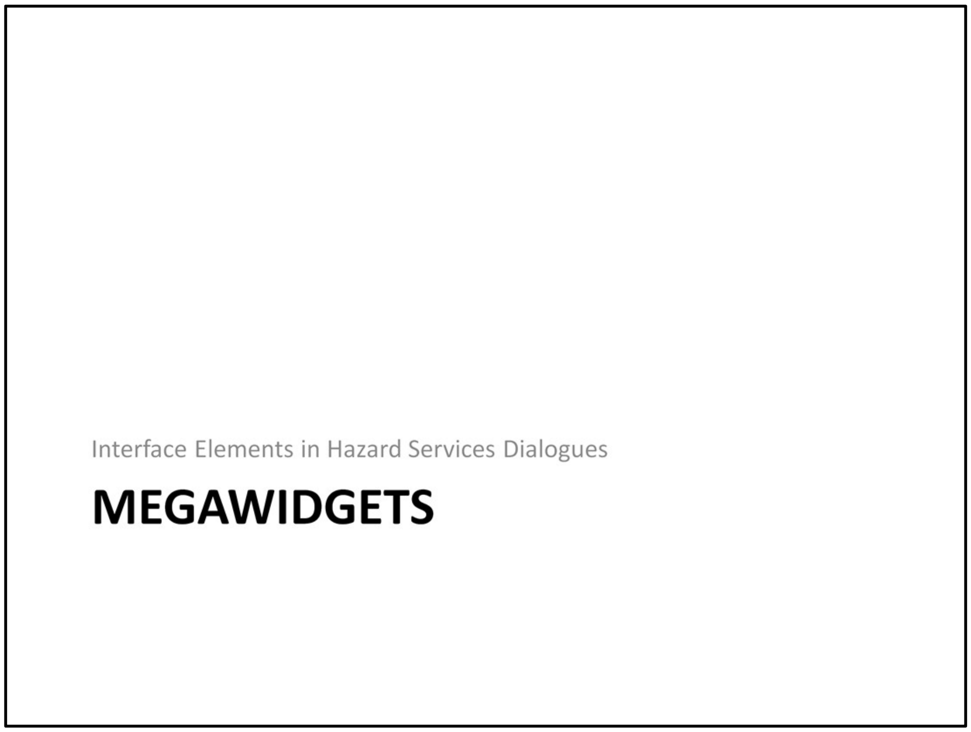
Before we transition to our second topic in this module, there's one further type of metadata file which, it could be said, blurs the line just slightly between hazard and product. These are the product staging metadata.

These files are very much in the minority, but are identifiable by the three-letter PILs in their filename, such as the FFA, FLW, and FLS seen in this example, in contrast to the typical structure seen in hazard-specific metadata of a two-letter phenomenon, followed by an underscore, followed by a one-letter significance. This inclusion of product identifiers in the filename signifies their unusual dependence on certain products to be triggered.

Now, while the overwhelming majority of metadata populate choices in Hazard Information Dialogue, these unique exceptions manifest in a second interface, called the Product Staging Dialogue, which appears AFTER the user has hit "preview" on the HID. We see a simple example of this with a River Flood Watch, using "Snow melt or Ice jam" as the immediate cause for demonstration purposes.

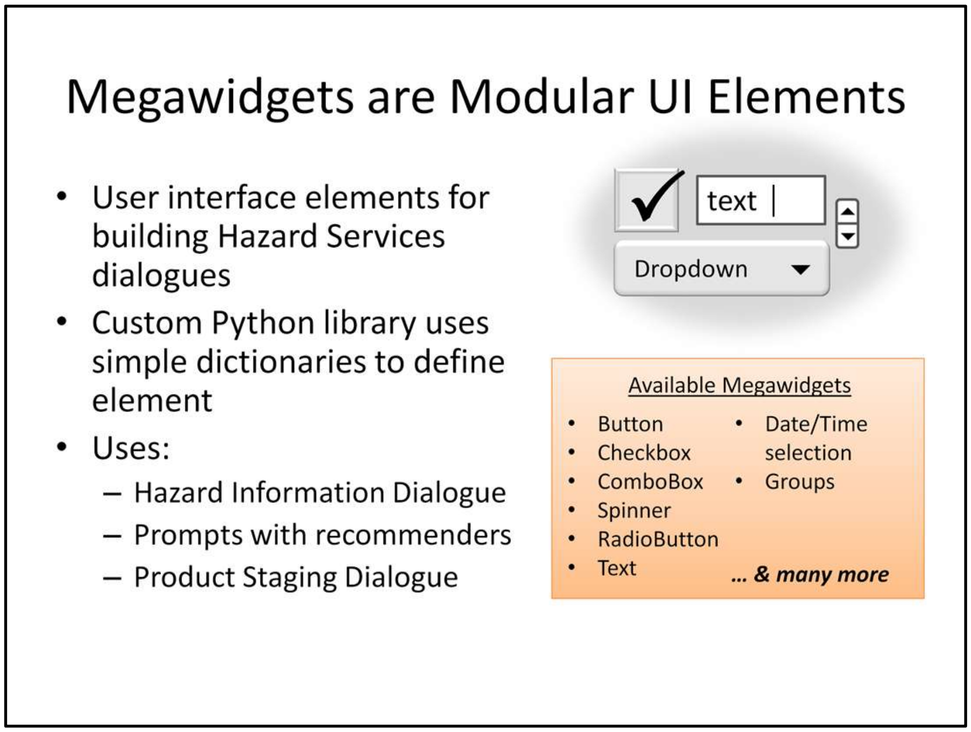
Without getting too distracted by this atypical variety, these specialized metadata serve a unique need where metadata-like attributes are determined to be needed by certain products. As product-serving attributes, they are only called upon if the relevant product is generated. They may include attributes similar to other metadata, such as call-to-action specifications, but which are intended to address the entire product.

But from a configuration standpoint, aside from the unique way in which they are called upon and presented, these product-staging metadata files share the same framework as regular metadata, in terms of their structure, dependence on Common Metadata, use of megawidgets, and so on.


Interface Elements in Hazard Services Dialogues

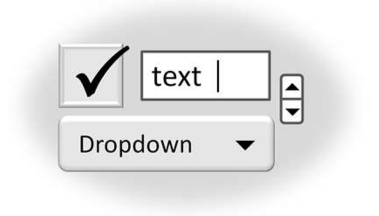
MEGAWIDGETS

In this next subsection, we briefly but formally introduce “megawidgets” and the importance they bear on Hazard Services interfaces.



Megawidgets are Modular UI Elements

- User interface elements for building Hazard Services dialogues
- Custom Python library uses simple dictionaries to define element
- Uses:
  - Hazard Information Dialogue
  - Prompts with recommenders
  - Product Staging Dialogue



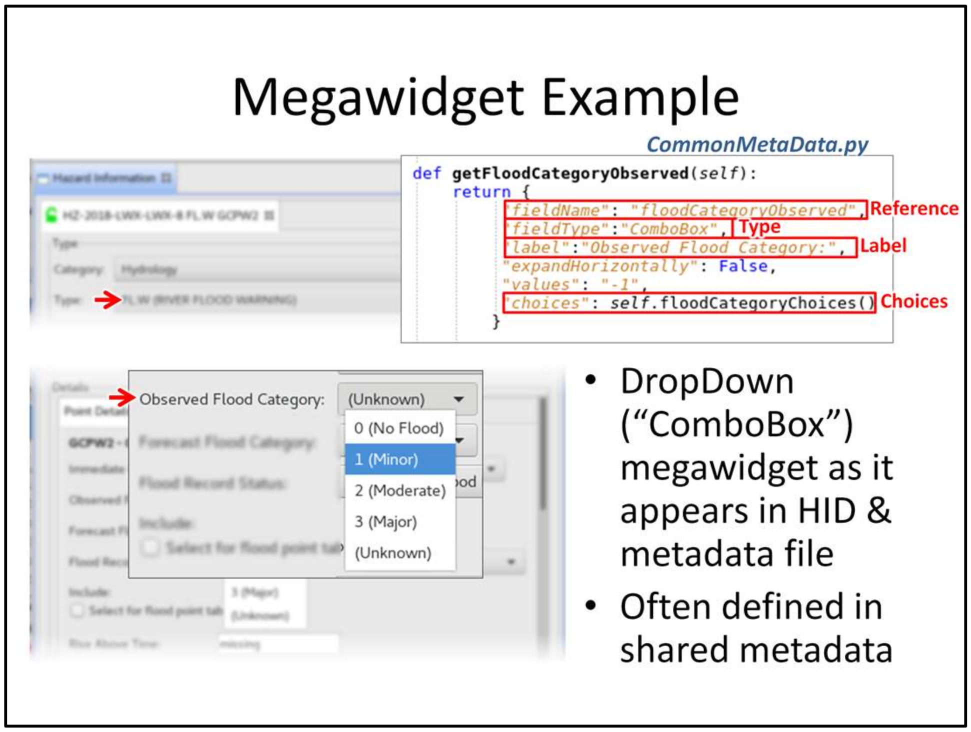
Available Megawidgets

- Button
- Checkbox
- ComboBox
- Spinner
- RadioButton
- Text
- Date/Time selection
- Groups

… & many more

The extensive customizability of Hazard Services is no less true of its displays. Megawidgets are the modular user interface elements used to construct these displays.

Designed to be fully editable through simple Python dictionaries, megawidgets are the building blocks which construct the Hazard Information Dialogue, as well as other prompts and dialogues like those which appear with recommenders and for product staging. The set of pre-built megawidgets available to use includes buttons, checkboxes, combo or drop-down boxes, and many more.



Megawidget Example

CommonMetaData.py

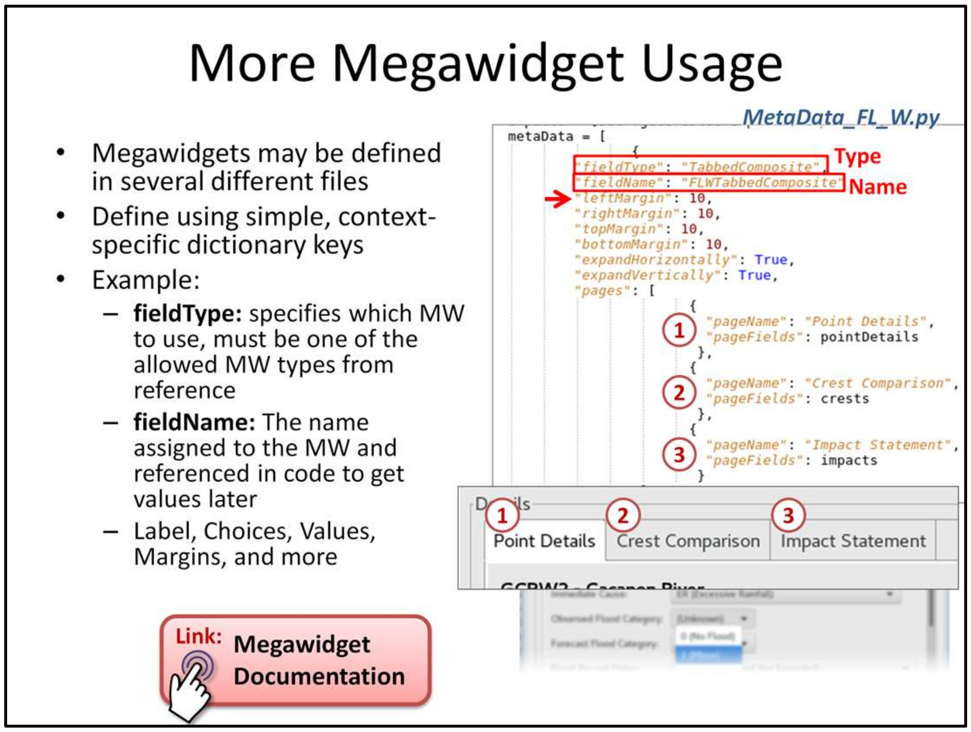
```
def getFloodCategoryObserved(self):
    return {
        "fieldName": "floodCategoryObserved",
        "fieldType":"ComboBox",
        "label":"Observed Flood Category:",
        "expandHorizontally": False,
        "values": "-1",
        "choices": self.floodCategoryChoices()
    }
```

Reference
Type
Label
Choices
Observed Flood Category:
(Unknown)
0 (No Flood)
1 (Minor)
2 (Moderate)
3 (Major)
(Unknown)

- DropDown (“ComboBox”) megawidget as it appears in HID & metadata file
- Often defined in shared metadata

The extensive customizability of Hazard Services is no less true of its displays. Megawidgets are the modular user interface elements used to construct these displays.

Designed to be fully editable through simple Python dictionaries, megawidgets are the building blocks which construct the Hazard Information Dialogue, as well as other prompts and dialogues like those which appear with recommenders and for product staging. The set of pre-built megawidgets available to use includes buttons, checkboxes, combo or drop-down boxes, and many more.



More Megawidget Usage

- Megawidgets may be defined in several different files
- Define using simple, context-specific dictionary keys
- Example:
  - **fieldType:** specifies which MW to use, must be one of the allowed MW types from reference
  - **fieldName:** The name assigned to the MW and referenced in code to get values later
  - Label, Choices, Values, Margins, and more

MetaData_FL_W.py

```
metaData = [
        {
        "fieldType": "TabbedComposite",
        "fieldName": "FLWTabbedComposite",
        "leftMargin": 10,
        "rightMargin": 10,
        "topMargin": 10,
        "bottomMargin": 10,
        "expandHorizontally": True,
        "expandVertically": True,
        "pages": [
                {
                "pageName": "Point Details",
                "pageFields": pointDetails
                },
                {
                "pageName": "Crest Comparison",
                "pageFields": crests
                },
                {
                "pageName": "Impact Statement",
                "pageFields": impacts
                }
```

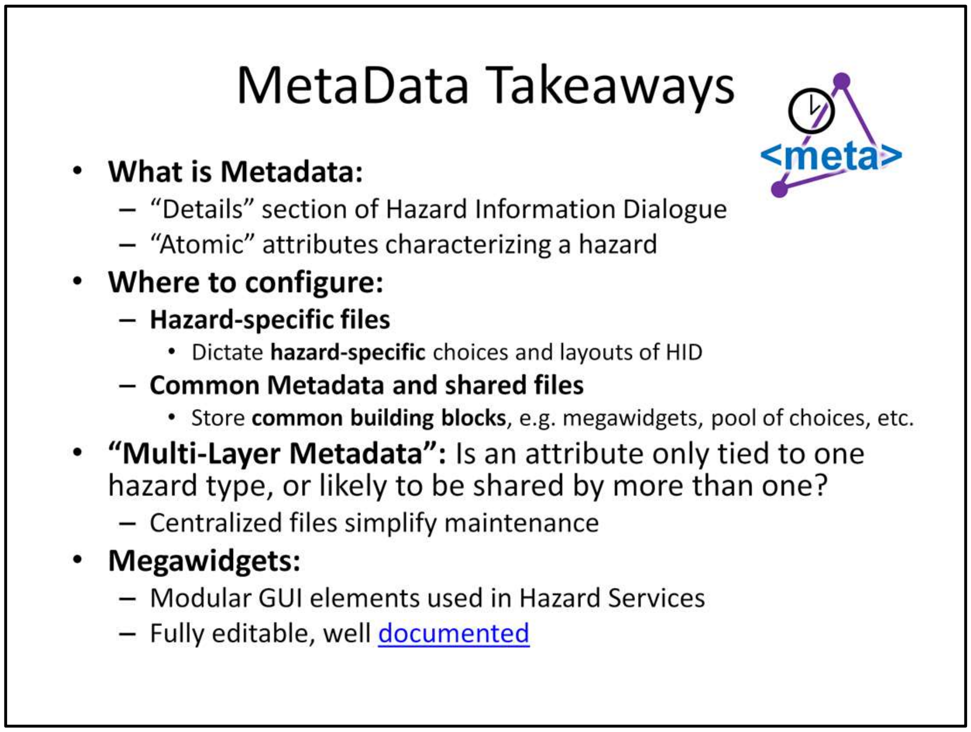
Link: **Megawidget Documentation**

Another example of a megawidget and its underlying code is shown here, this time one which creates tabbed “pages” on the HID, useful for organizing other groups of megawidgets. We’re using the same FL.W Hazard Information Dialogue as before for this example.

Although we’ve emphasized that, for practicality, many megawidgets are defined in commonMetadata, they can certainly be specified in other files, and in fact the megawidget framework is used in many other components of Hazard Services to create dialogues. In this case, we see the tab format specifically defined in the hazard-specific Metadata for FL.W. Evidently, the particular design and choice of tabs in this layout is specific enough to FL.W that it’s sensible to store its setup within that file.

The megawidget toolkit was designed for Hazard Services to enable simple specification of interface elements through straightforward python dictionaries. By specifying the field Type, field name (which again is needed within code to get its value later), and a handful of other fields (determined by checking the requirements for each megawidget in GSD’s documentation), Focal Points have extensive control over how the Hazard Information Display and other dialogues look and behave. In the case of this “TabbedComposite” megawidget, we see that a special sub-dictionary for pages is necessary, with pageNames and fields for each tab, again outlined clearly in the megawidget reference.

Due to their importance to Hazard Services, and the likelihood that focal points may want to tweak their behavior or otherwise alter megawidgets in the HID, focal points are encouraged to refer to a comprehensive document on megawidgets provided by GSD in the references.



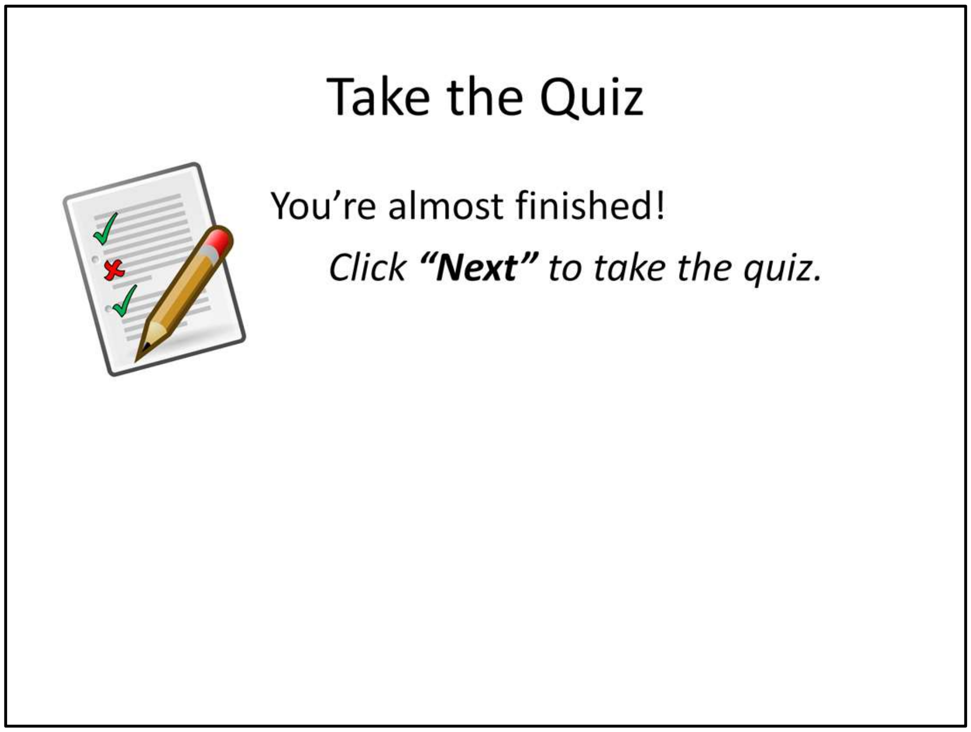
MetaData Takeaways

- **What is Metadata:**
  - "Details" section of Hazard Information Dialogue
  - "Atomic" attributes characterizing a hazard
- **Where to configure:**
  - **Hazard-specific files**
    - Dictate **hazard-specific** choices and layouts of HID
  - **Common Metadata and shared files**
    - Store **common building blocks**, e.g. megawidgets, pool of choices, etc.
- **"Multi-Layer Metadata":** Is an attribute only tied to one hazard type, or likely to be shared by more than one?
  - Centralized files simplify maintenance
- **Megawidgets:**
  - Modular GUI elements used in Hazard Services
  - Fully editable, well documented

In this section we've covered Hazard Metadata, and their inherent relation to the Hazard Information Dialogue. Hazard Metadata are generally unformatted, product-independent attributes which characterize the nature of a hazard, and owing to the typical differences between attributes of one hazard type and another, metadata configurations are most often, understandably, separated into different, hazard-specific files. These files directly dictate the assembly and choices available in the HID for that hazard type.

But we also saw that there's more to metadata than just hazard-specific files. Despite differences in their assembly, a respectable degree of similarity in the building blocks of the HID, such as megawidgets and the choices therein, make it practical to specify common elements in a shared, parent metadata file, resulting in a multi-layered metadata structure. This use of centralized files, while maybe uncomfortable at the outset, concentrates a lot of useful functionality in just one place for dependent files to use, ultimately greatly simplifying maintenance.

Finally we've covered megawidgets and their importance in assembling Hazard Services' interactive interfaces. Megawidgets are a custom Python library implemented for Hazard Services, which enables simple Python dictionaries with standardized language to create a wide variety of complex and useful interface elements, including checkboxes, menus, tab groupings, and much more. By referring to GSD's thorough documentation on the necessary parameters for each megawidget, focal points will be able to customize the pieces which make up the interactive dialogues in Hazard Services.

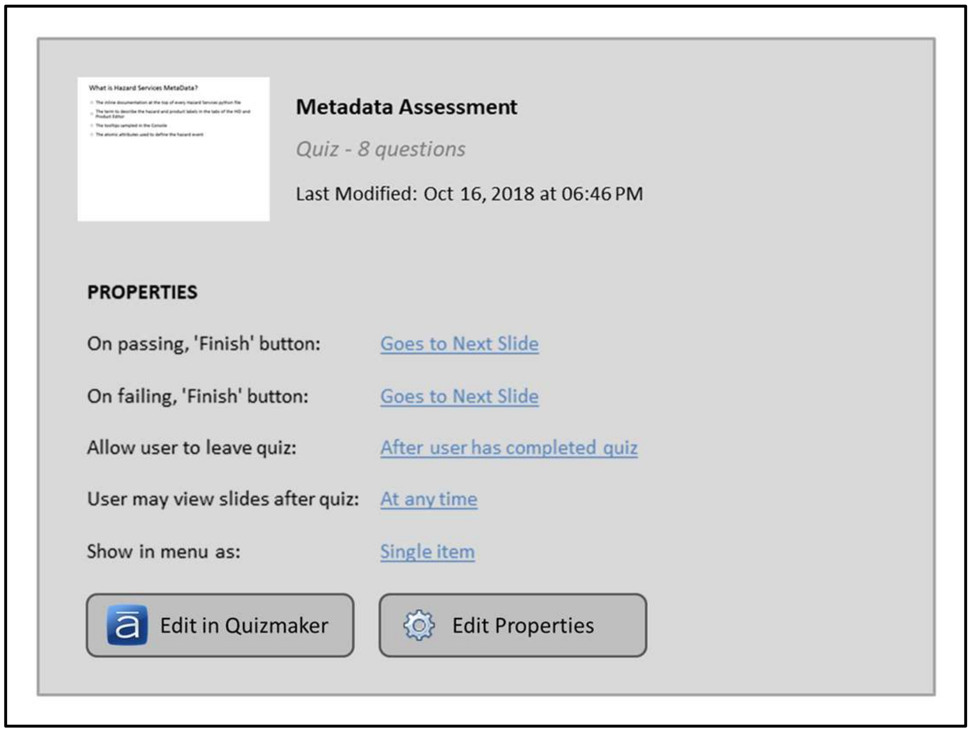


Take the Quiz

You're almost finished!

*Click **"Next"** to take the quiz.*

[No audio for this slide]



What is Hazard Services MetaData?

- The inline documentation at the top of every Hazard Services python file
- The term to describe the hazard and product labels in the tabs of the HID and Product Editor
- The tooltips sampled in the Console
- The atomic attributes used to define the hazard event

Metadata Assessment

Quiz - 8 questions

Last Modified: Oct 16, 2018 at 06:46 PM

PROPERTIES

On passing, 'Finish' button:	Goes to Next Slide
On failing, 'Finish' button:	Goes to Next Slide
Allow user to leave quiz:	After user has completed quiz
User may view slides after quiz:	At any time
Show in menu as:	Single item