Working with Elements

Bentley®

Agenda

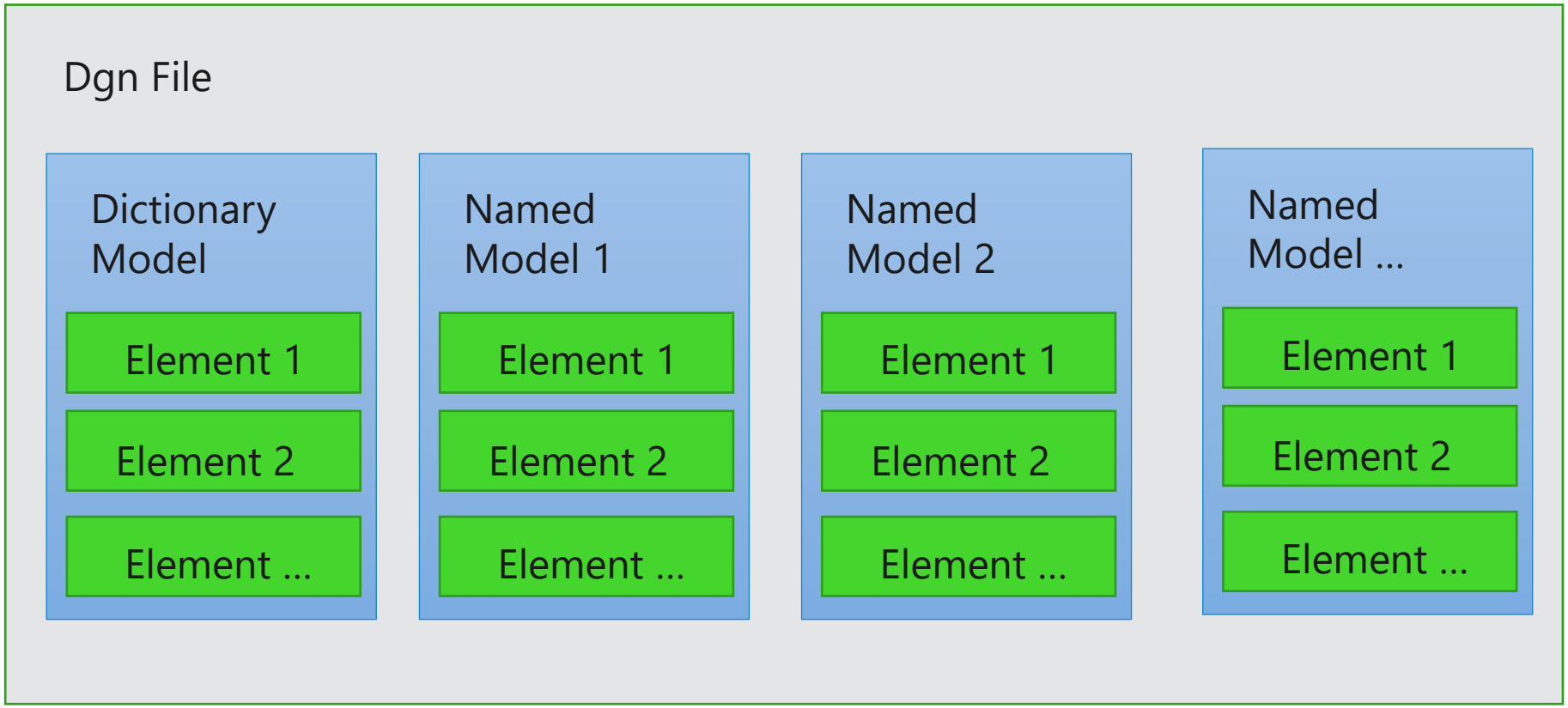
- DgnFile, DgnModel and Elements
- Elements
  - ElementHandle
  - EditElementHandle
  - Complex Elements
  - MSElementDescr
  - ElementHandler
  - ElementAgend
- Storing Data on Elements
  - Xattribute
  - Linkages

Agenda

- Dependency Management
  - Persistent Element Path (PEP)
  - Dependency Manager
  - Change Propogation
- Exercises

DgnFile, DgnModel and Elements

- A Dgn file is a collection of DgnModels
- A Dgn Model is a collection of Elements



Bentley

Elements

- Elements are basic building blocks of a dgn file
- All information stored in a dgn file is stored in Elements
- There are two types of elements
  - Non-Graphical Elements:
    - Non-Graphical elements are never displayed, they are used purely for storing information in dgn file
    - Eg: NamedGroup element, used for grouping bunch of elements.
  - Graphical Elements:
    - Graphical elements are displayed by the view
    - Eg: Shape element, used for storing data of a 3D Shape

ElementHandle

- An ElementHandle provides readonly access to the underlying element, it represents elements current state
- ElementHandle is also a very convenient way to access Elements from a Model for read only access
- ElementHandles are lightweight objects and are usually stack-based
- ElementHandles can be constructed from either an ElementRef and ModelRef

ElementHandle

```
pathCurve = ICurvePathQuery.ElementToCurveVector (eh)
```


EditElementHandle

- EditElementHandle is the ideal means of passing an element to functions, when the fun ctions might modify the element.
- EditElementHandle is also useful for creating and adding new elements and for deletin g existing elements.
- EditElementHandle can hold either an ElementRefP or an MSElementDescrP in order to represent an element
- You can use EditElementHandle to add, replace, or delete an element's XAttributes

EditElementHandle

```
point = DPoint3d (0.0, 0.0, 0.0)

eehEllipse = EditElementHandle ()

status = EllipseHandler.CreateEllipseElement (eehEllipse, None, point,
math.pi/6, math.pi/7, 0.0 , False, modelRef)

eehEllipse.AddToModel()
```


Bentley

References

Help (MSPyDgnPlatform.EditElementHandle)

- Documentation of EditElementHandle from PowerPlatform SDK

Bentley

Complex Elements

- Represent hierarchy of elements
- Parent Element is called Complex Header element
- Child elements are called complex componets
- Ex: Cell Element

Complex Element Example

```
NUM_LINES = 3
lines0 = EditElementHandle()
lines1 = EditElementHandle()
lines2 = EditElementHandle()
Lines = [lines0, lines1, lines2]
m_points = [[DPoint3d() for i in range(2)] for j in range(NUM_LINES)]
ChainHeaderHandler.CreateChainHeaderElement(m_eeh, None, True, modelRef.Is3d(), modelRef)
 for j in range(0, NUM_LINES):
    pt1 = DPoint3d(m_points[j][0].x, m_points[j][0].y, m_points[j][0].z)
    pt2 = DPoint3d(m_points[j][1].x, m_points[j][1].y, m_points[j][1].z)
    seg = DSegment3d(pt1, pt2)
    LineHandler.CreateLineElement(Lines[j], None, seg, modelRef. Is3d(), modelRef.)
    ChainHeaderHandler.AddComponentElement (m_eeh, Lines[j])

ChainHeaderHandler.AddComponentComplete(m_eeh)
```


ElementHandler

- ElementHandlers provide behavior to Elements in MicroStation
- ElementHandler defines the standard queries and operations available on all Elements
- Every element in MicroStation has a ElementHandler
- There is only one ElementHandler per Element type
- Given an ElementHandle or ElementRef its ElementHandler provides a way to quickly check the type of the element

Element Agenda

- A vector of EditElementHandle entries to be used for operating on groups of elements
- ElementAgendas are created by MicroStation from sources that work on multiple elements such as the SelectionSet and Fences

ElementAgenda

```
ag =  ElementAgenda ()
ag.Insert (ref1, model)
ag.Insert (ref2, model)
```

… NEEDS WORK

Bentley

Data Storage on Elements

- Every Element can have additional data stored on to them

Eg: an Element might want to store its co-ordinates or a string identifier such as name.

- Ways of storing data on an Element
  - Linkages
  - XAttributes

Linkages

- Linkage are pieces of data that can be stored on elements
- Every Linkage stored on an Element can be identified by a unique Linkage id
- Linkage Id's must be created and distributed by Bentley only to stop clashing with other vendor Id's
- Data stored on a Linkage has limitation of data size since Linkages are stored in continuous memory location
- Usage of any new data on an element using Linkages is highly discouraged, please use Xattributes instead
- Linkage is stored in Element

XAttributes

- An XAttribute is information that can be stored on an Element
- There can be more than one XAttributes stored on an Element
- There is no size limit on XAttributes
- An Element knows about all of its XAttributes and XAttributes know about their Element
- Xattribute is stored on element
- An XAttribute can be identified by a unique UInt64 identifier which consists of
  - An XAttribute Handler Id 32 bits
    - XAttribute Minor Id 16 bits
    - XAttribute Major Id 16 bits
  - An Xattribute Id 32 bits
- The Xattriubte Major Id must be provided by Bentley to avoid collisions with other vendors

PersistentElementPath (PEP)

- Defined in MSPyDgnPlatform
- Captures the reference to an element in the form of a path
  - Path starts from the dependent element
    - Dependent element is contained in home model
  - Path's destination is the root element
    - Root element is contained in root model
- Generally used by an XAttribute that wants to store a pointer to an element
- Can also be used in other contexts
- More complex PEP functionality relating to behaviors on copy is only available to C++

Types of PEPs

- Same or different Models
  - Dependent & root element in same model
  - Dependent & root element in different models or files
- Root can simple or complex
  - Root is a simple (non-complex) element
  - Root is part of a shared cell

Bentley

PEP Constructors

- To create a PersistentElementPath, need to specify the root model & element
- PersistentElementPath (DisplayPathCP, DgnAttachmentCP)
  - Constructor to DisplayPath: an element & possibly its sub-component
  - DgnAttachmentCP – refers to an optional primary parent attachment
- PersistentElementPath(DgnModelRefP, ElementRefP, DgnAttachmentCP)
  - Convenience constructor. Equivalent to above constructor as follows:
    - DisplayPath dp (ElementRef,P DgnModelRefP)
    - PersistentElementPath (dp, DgnAttachemntCP)
- PersistentElementPath (ElementRefP)
  - Constructor to a root element in same model as dependent element
- PersistentElementPath (DgnAttachmentCP)
  - Constructor to a reference attachment

PEP Evaluate Methods

- To evaluate a PEP, need to specify the home model, the start of the path.
- Evaulate to root element
  - EvaluateElement (): return root element in EditElementHandle
  - EvaluateElementFromHost (): return root element in EditElementHandle
  - EvaluateElementRef (): return root element in ElementRef
  - GetDisplayPath (): return root element in DisplayPath
- DisclosePointers () : reports all elements referenced by PEP (or all element in path to root element)
- DependsOnElementRef (): Check if PEP depends on specified ElementRef
- EqualElementRef (): Check if specific elementRef is the root element
- ProcessPath (): Process items in a PEP

PEP & XAttribute Persistence

- PEP is normally embedded in an XAttribute
- Save & Load of XAttribute with PEP
  - Call PEP.Store () on each PEP in XAttribute – when XAttribute is saved
  - Call PEP.Load () to restore PEP state from raw (persistent) data stored in XAttribute
- ToString ()
  - Serialize state of PEP in form of a string
- FromString ()
  - Recreate PEP from stored state string

PEP & XAttribute Copy

- When an element is copied, then its XAttributes are also copied.
- If XAttribute points to another element then the handler controls how the PEP is affected
- Need to decide if the root element needs to be deep copied or not. If so, then remap of IDs is required

Bentley

PEP Copy Options

- What to do when the dependent element is copied?
- eCOPYOPTION_RemapRootsWithinSelection
  - If the root is also being copied, then remap the pointer of the dependent's copy to the root's copy
  - If the root is not being copied, then set mark dependent's PEP as invalid
- eCOPYOPTION_DeepCopyAcrossFiles
  - Graphic root: same as RemapRootsWithinSelection
  - Non-Graphic root (root is in control section of model)
    - Deep-copy root when copying between models & remap pointer of dependent's copy to the root's copy
    - Dependent's copy points to original root when copying within same model
  - Dictionary root:
    - Deep-copy root when copying between files & remap pointer of dependent's copy to the root's copy
    - Dependent's copy points to original root when copying with same file

PEP Copy Options

- eCOPYCONTEXT_DeepCopyRootsAlways
  - Graphic root: same as RemapRootWithinSelection
  - Non-Graphic root:
  - Dictionary root:
    - Always deep copy root & remap pointer of dependent's copy to the root's copy
- eCOPYOPTION_PreserveReferences
  - If PEP contains a DgnAttachment, then preserve the reference of dependent's copy to the DgnAttachment (or to the copied DgnAttachment)
  - IF PEP does not contain a DgnAttachment then same as RemapRootWithinSelection

DependencyLinkage

- Linkage which stores dependency related information
- Old Style
- Consists of
  - An array of element Ids – represents the root element
  - An AppId – each unique and provided by Bentley
  - An AppValue – value stored by application
  - One or more AssocPoints - an AssocPoint represents a type of association – similar to PSP
  - CopyOptions – similar to PEP Copy Options

Dependency Manager

- Keeps track of changes to root elements and notifies dependent elements
  - Called Change Propagation
- Also collaborates with CopyContext to remap dependencies
- Typically notification occurs automatically in the context of a transaction
  - Itxn.ValidateCurrentTxn triggers change propagation
- Application can invoke change propagation on demand

Change Propagation

- The process of notifying a dependent element that its root element has changed
- Occurs in multiple phases
  - A dependent element may be a root element for another dependent element
  - Thus if a first level dependent element changes when its root changes, then the second level dependent element needs its own notification
  - Circular change propagation is handled by aborting the change propagation after a certain number of cycles
  - Assumption is the circular change propagation dampens within a certain number of cycles.

Key Dependency Manager Methods

- All methods are static
- ProcessAffected ()
  - Invokes change propagation. Essentially invokes callbacks on all dependent elements whose root elements have changed.
  - Also in performs dependency pointer remapping for copied dependent elements

Supporting Dependency Manager Methods

- RootChanged ()
  - Declare that an element changed, even if it did not really change
  - If an element really changed, then typically Dependency Manager will pick change automatically
- IsInCurrentChangeSet ()
  - Check if specified elementRef is affected by root changes in current phase of change propagation
- WasAdded ()
  - Check if an element was added since the last time ProcessAffected () was called
- SetTraceLevel () – for debugging only

Connect with Bentley

Connect with us

www.bentley.com

developer.bentley.com

communities.bentley.com/products/programming

Social Media

Medium.com

MicroStation

