BD 35-3 # Building Stronger Families

David Conant – Revit Product Designer
Steve Stafford - WATG

- Adding your own custom data to families
- Effective use of formulas
- Conditionals in formulas
- Controlling nested families

Building Stronger Families

Agenda

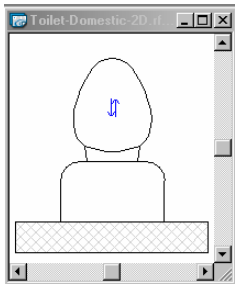
- Family Planning
  - Good planning strategies
    - Complexity
    - Typology
    - Constraints
  - Test for Success
- Learning to Share
  - Creating Shared Parameters
  - Effective Use of Sharing
- Remember Your Math
  - Formulas in families
  - Syntax and Methods
  - Examples
- Decision Making
  - Conditional Statements in Families
  - Syntax and methods
  - Limitations
  - Examples
- Good References
  - Using Reference Lines to control component orientation
- Controlling the Nest
  - Using Family Type parameters
  - Scheduling nested family types

Family Planning

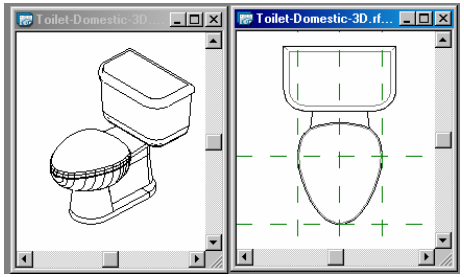
- Pre-construction planning makes for better and more easily constructed families
  - How will the family be used?
    - What views?
    - Levels of Detail
    - Proper Category
    - Is interactive resize required?
  - Understand the typology
    - Unique type for each variation
    - Many variants within one type
    - Mix
  - Determine the parameter scheme
    - What are the realistic variations
    - What values will need to appear on schedules
    - What values will need to appear in exports

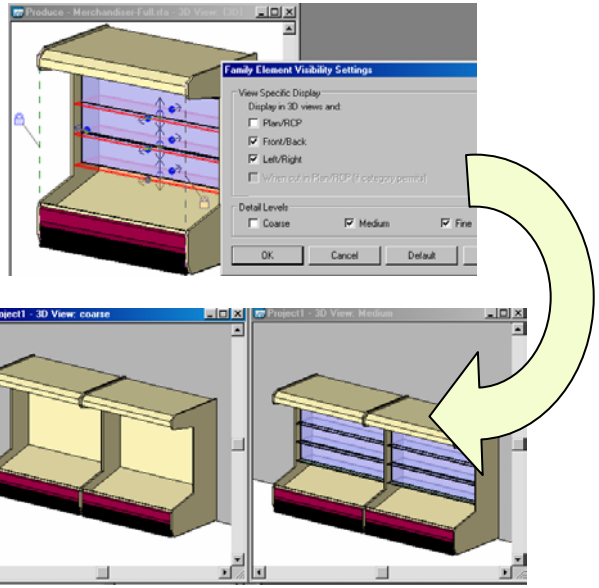
Build For Speed

- Use the simplest representation possible or you will pay a price in perfomance
  - If it can't be seen don't build it
  - If you don't need to show it as a solid in a 3d view, build it as 2d elements only.
  - Avoid modeling tiny things.
  - If 3d detail can't be seen in a typical view, simplify it.
  - If detail is important in some views, assign level of detail visibility to elements to hide them when not needed.



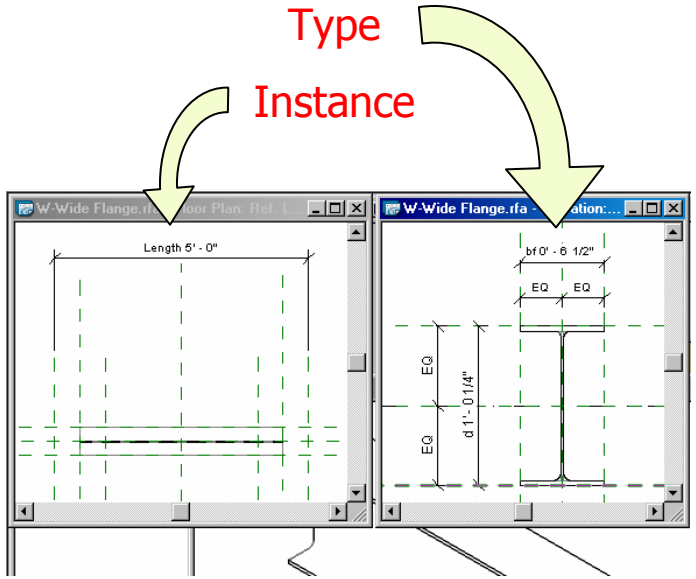
VS





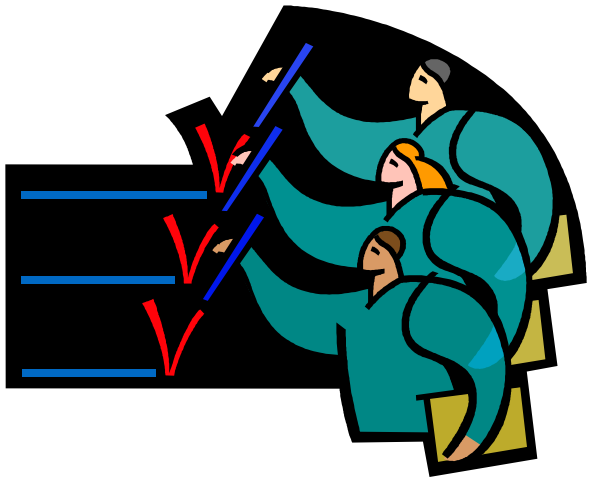
Put Your Family on its Best Behavior

- Constraints and Parameters add Control but…
  - Don't go overboard
    - Excessive constraint usage hurts performance, and makes families harder for others to understand.
    - Too many parameters make variation more difficult
  - Instance Parameters allow variation within a type, but…
    - Should only be used when an element is truly freely resizable in the real world
    - Each variation of size creates a unique symbol. This decreases performance
  - Shared Parameters expand access to your data, but…
    - Parameters from different shared parameter files will always be different even if they have the same name.



Put Your Family on its Best Behavior

- Standardized Testing Ensures Quality
  - Flex the skeleton before adding geometry
    - Fixing a bad skeleton is far easier than rebuilding geometry
  - Flex repeatedly during construction
    - Discover problems while they are still small
  - Flex host thickness to see if all elements behave.
  - Be unreasonable to find the limits
    - Never expect users to behave reasonably
    - Always test at least 2x the largest and .5x the smallest reasonable values
  - Test visibility of all elements
  - Load into a project and try to make all types

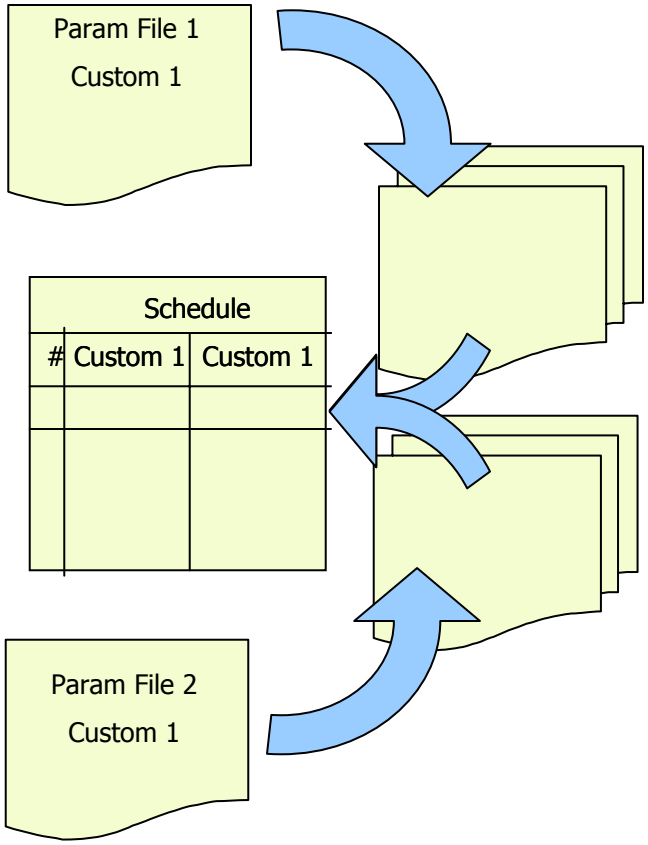


Learning to Share: Align your Data

- Shared Parameters align data across families and projects
  - Shared parameters can appear on all schedules
  - Shared Parameters allow elements of different categories to appear in the same schedule
  - Shared parameters can be tagged
  - Shared Parameters export to ODBC

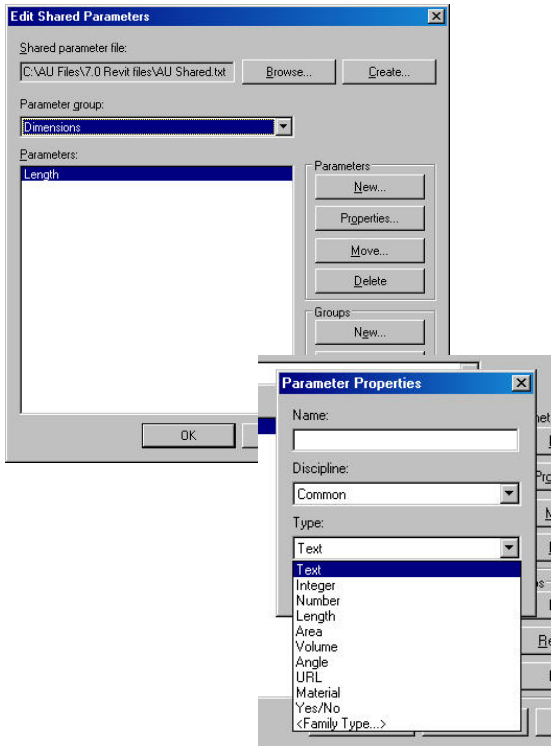
Learning to Share: Maintaining Discipline

- With power comes the potential for confusion
- Create a single shared parameter file for all users to access
  - Shared parameters get a unique ID on creation
  - Parameters with the same name created in different shared parameter files have different ID's and **<u>will not merge</u>**
  - Duplicate parameters create duplicate schedule entries
  - If duplication occurs, use Modify in family properties to re-associate all duplicates to one original



Learning to Share: Creating Shared Parameters

- Define and Modify:
  - Select a Shared Parameter File
    - **Best Practice:** Use only 1 shared parameter file for your firm
  - Select or Create a Parameter Group
    - Organize parameters into logical groups
  - Create a new Parameter
    - Name
    - Discipline
      - Use "Common"
    - Type
  - **Warning:** Once created, shared parameters cannot be modified or renamed.



Learning to Share: Applying Shared Parameters

- As Family Parameters

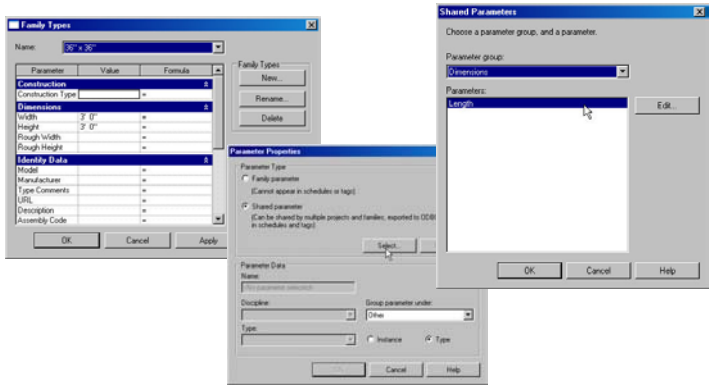
Use where values are known in advance.

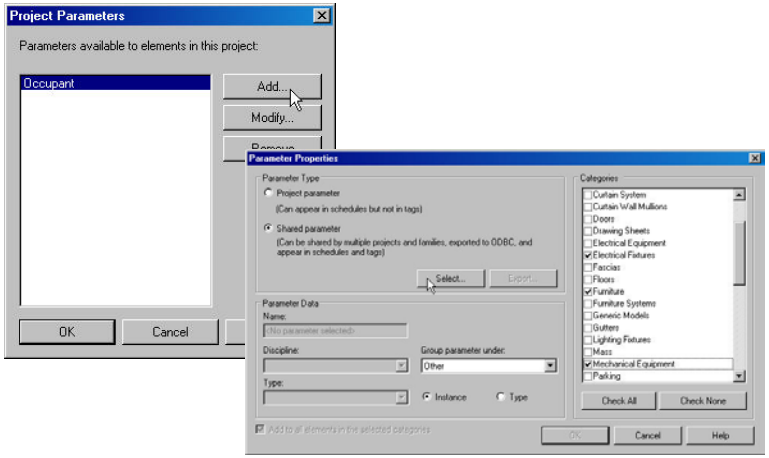
  - In Family Properties select Add
  - Select "Shared Parameter" from Parameter Properties

- As Project Parameters

Add shared parameters to all elements of a category in a project.

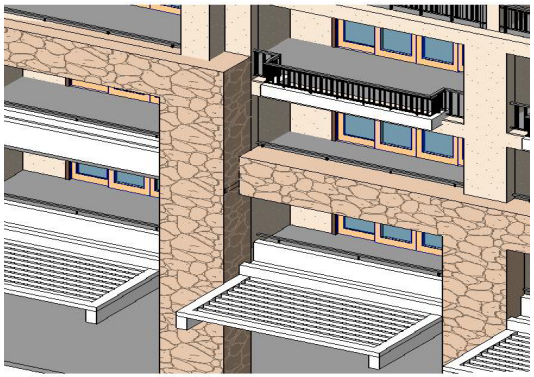
  - Usage:
    - •System families
    - •Sheets and views
    - •Where parameter value is unknown in advance.
  - Available from Settings|Project Parameters
  - Assign to desired categories

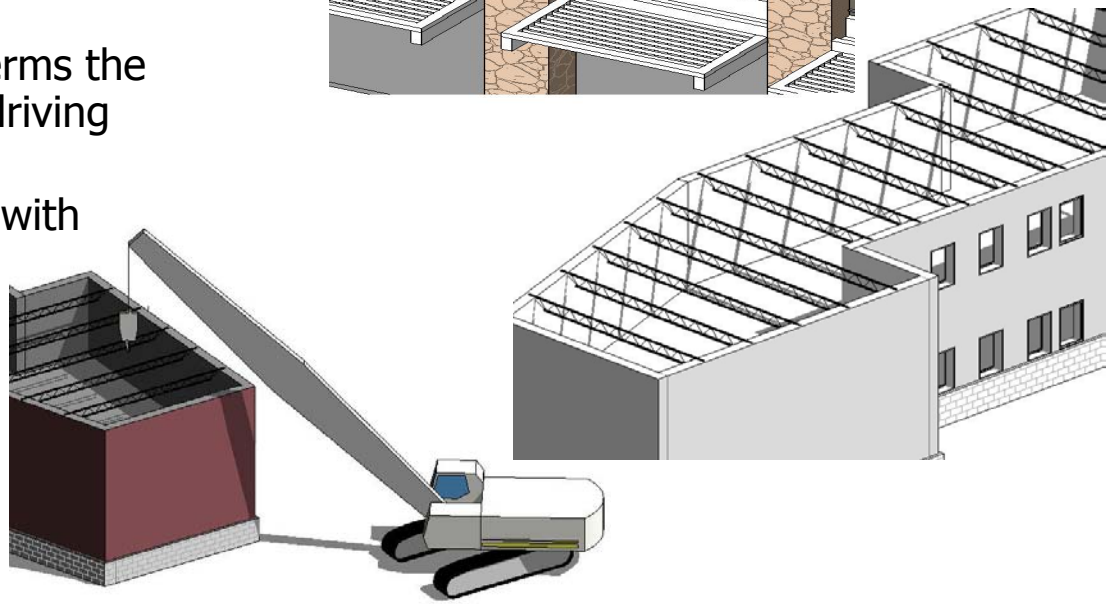




The Power of Formulas

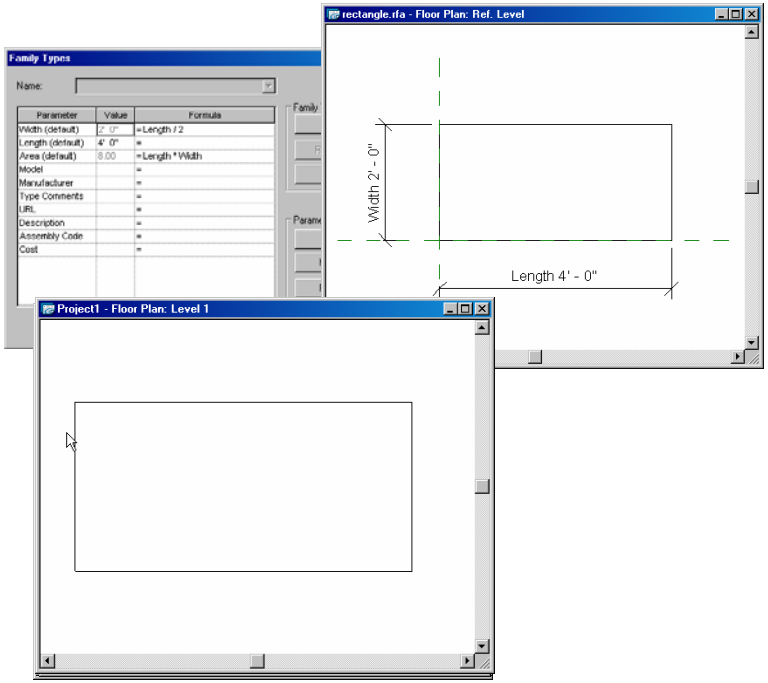
- Why? I hate math.
  - Create parameters that depend on other parameters
  - Embed fixed relationships more complex than simple constraints such as equality allow
  - Express parameters in terms the user understands while driving more complex criteria
  - Simplify user interaction with family parameters





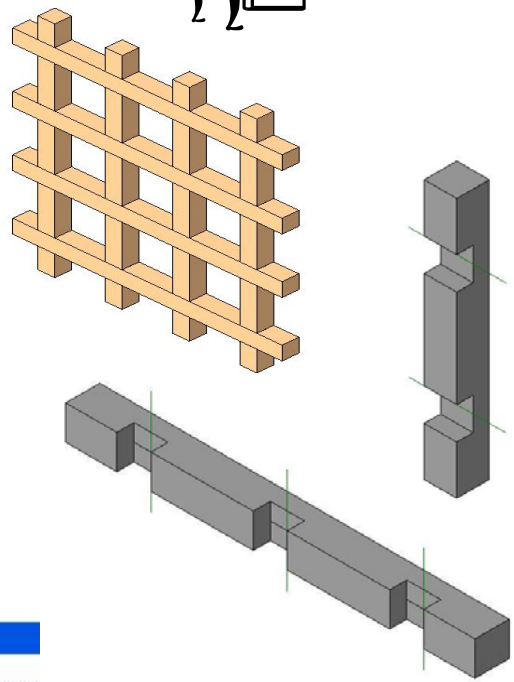
The Power of Formulas

- Data Types
  - Formulas can create real numbers, integers, lengths, areas, volumes, or angles
  - Multiply or divide by a single unit to convert between lengths, areas, reals, or integers.
- One way and two way formulas
  - If a parameter depends on only one other, changing either will change the other
  - If a parameter a function of more than one other parameter, it will be completely dependent.



Remember Your Math?

- How?
  - Enter a mathematical formula in the Formula Column of the Family Types dialog.
- Operators
  - +, -, /, *, sqrt, log, x^y, sin, cos, tan, atan, asin, acos, exp, abs
- Format
  - Use normal mathematical syntax
  - Parameter names are case sensitive
- Length = Height + Width + sqrt(Height*Width)
- ArrayNum = Length/Spacing



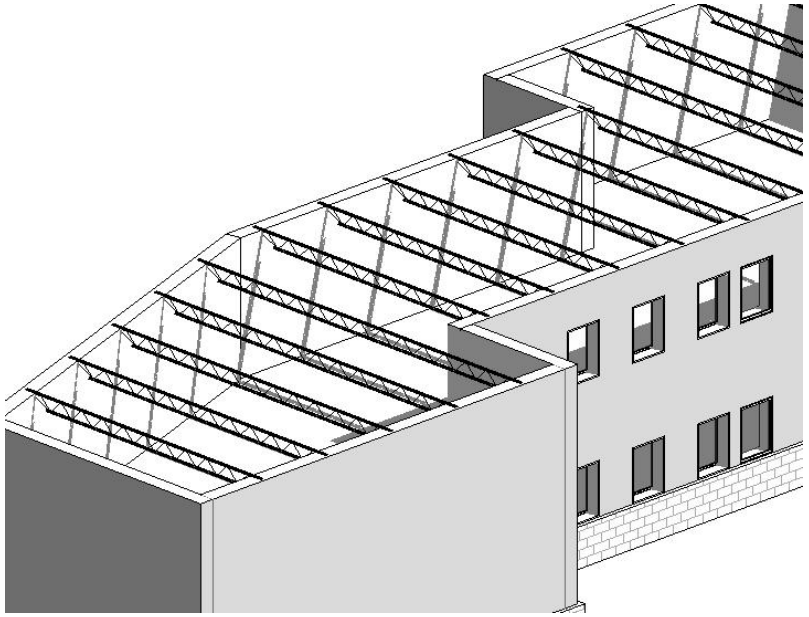
Other		
Timber Width	200.0	=
Timber Notch Spacing	800.0	=
Timber Overall Length	2400.0	= (2 * Timber Notch Start) + Timber Notch Spacing * (Timber Array Count - 1)
Timber Notch Start	400.0	
Timber Material	<By Cate	=
Timber Array Count	3	=

Instance Parameters in Formulas

- Instance parameters can be driven by either type or instance parameters
- Instance parameters cannot drive type parameters
- An instance parameter that depends on another instance parameter is a one way parameter. It will update with each change in its driver

New in 7.0

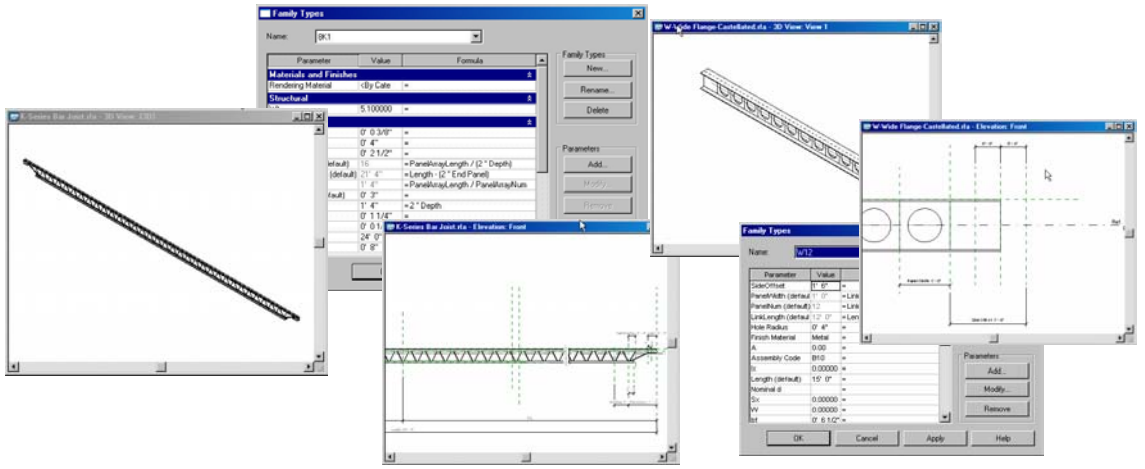
- Beam length from 2 pick beams can be used in formulas



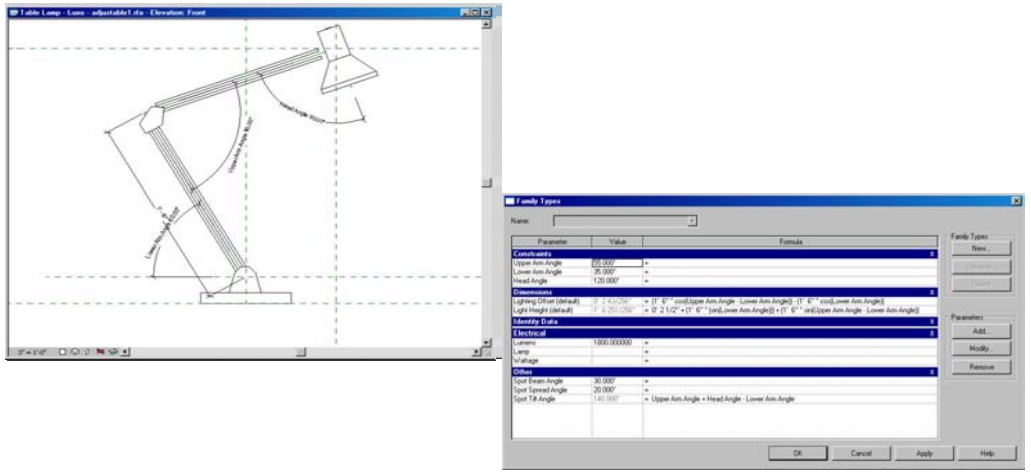
Examples

Examples

- Arrays

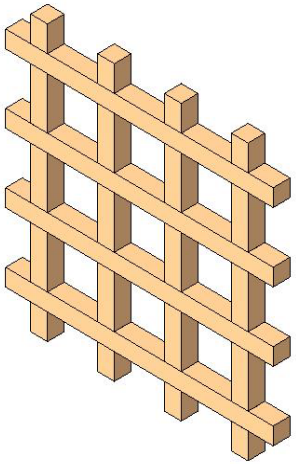


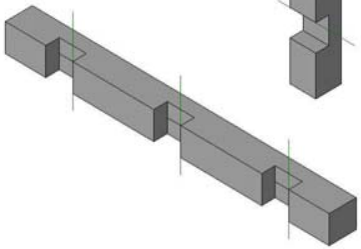
- Angles

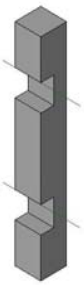


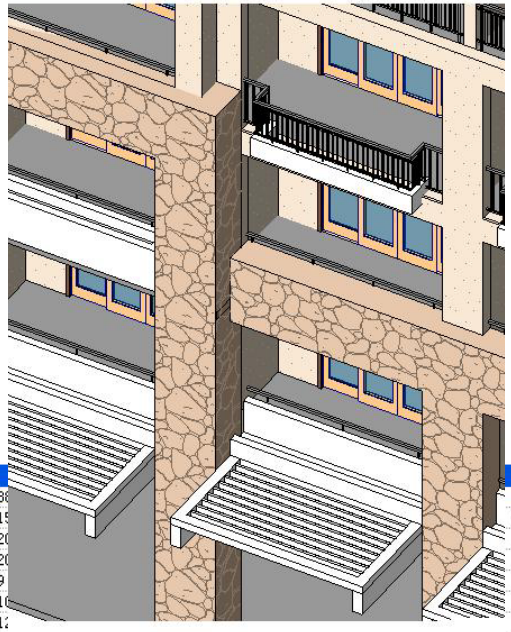
Formulas: Examples from Practice

Other		
Timber Width	200.0	=
Timber Notch Spacing	800.0	=
Timber Overall Length	2400.0	= (2 * Timber Notch Start) + Timber Notch Spacing * (Timber Array Count - 1)
Timber Notch Start	400.0	=
Timber Material	<By Cate	=
Timber Array Count	3	=







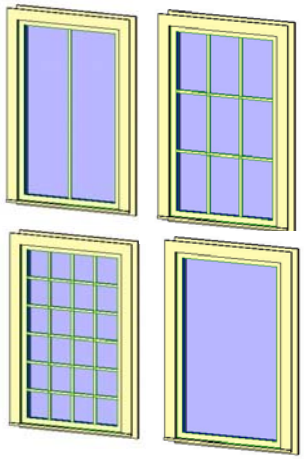


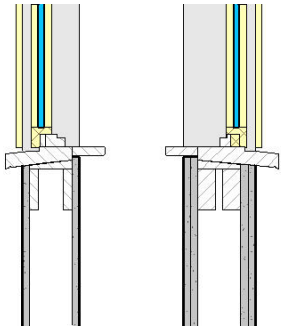
Parameter	Value
Graphics	
Timber Material	Wood - Cherry
Array Vertical	4
Array Horizontal	4
Dimensions	
Timber Width	200.0
Timber Notch Start	400.0
Timber Spacing	800.0
Identity Data	

Other		
Planter Width Overall	3	
Planter Wall Thickness	1	
Planter Slab Offset	2	
Planter Screen Frame Thickness	2	
Planter Screen Fin Count	9	
Planter Railing Height	1	
Planter Railing Clearance	1	
Planter Material Fin	<By Category>	
Planter Material Beam	<By Category>	=
Planter Floor to Floor Height	4900.0	=
Planter Floor Thickness	150.0	=
Planter Fin Width	50.0	=
Planter Fin Length	3400.0	= Planter Width Overall - (2 * Planter Beam Width)
Planter Fin Height	120.0	= (Planter Bottom of Beam - Planter Beam Height) - 30 mm
Planter Fascia Height	350.0	=
Planter Extension	2000.0	=
Planter Bottom of Beam	600.0	= Planter Beam Height + Planter Floor Thickness
Planter Beam Width	200.0	=
Planter Beam Height	450.0	=
Planter Beam Attachment Depth	600.0	=
Planter Fin First Offset	150.0	=

Decision Making: Conditional Formulas

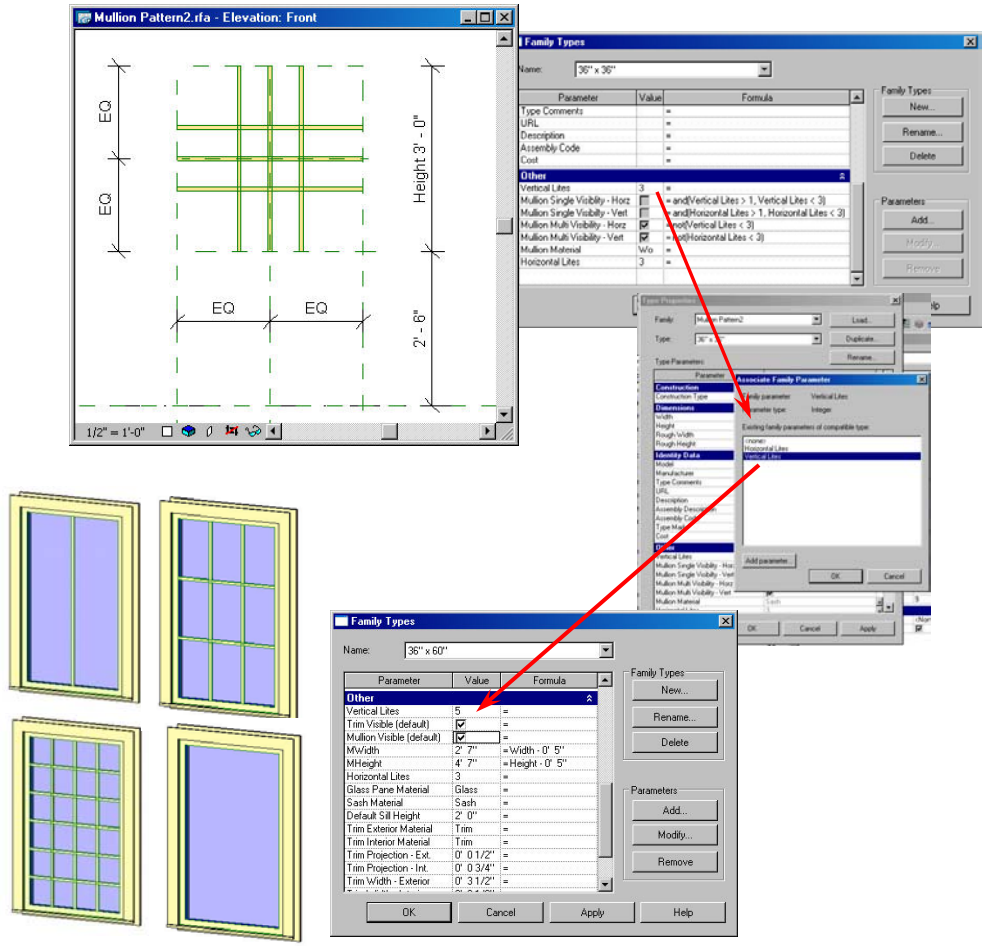
- Add logic to family behavior
  - Define actions in a family that depend on the state of other parameters
- Why Bother?
  - Non-Linear size variation
    - IF this beam is longer than 20’ make the depth 16”, otherwise make it 12”.
  - Conditional Visibility
    - Is a statement True? If so, make an element invisible.
  - Limit Array Values
    - If the Array number is less than 2 reset it to 2





Decision Making: Conditional Formulas

- Logic and Scope
  - Conditional formulas are based on evaluation of a statement for truth
  - Only mathematical statements or YES/NO parameters can be tested.
  - Only mathematical statements or YES/NO parameters can be driven
  - Conditions in nested elements are hidden in the host
- Functions and Tests
  - Equal: =
  - Less Than: <
  - Greater Than: >
  - IF
  - NOT
  - AND
  - OR
  - <= and >= can be written as NOT(a > b).



Decision Making: Conditional Formulas

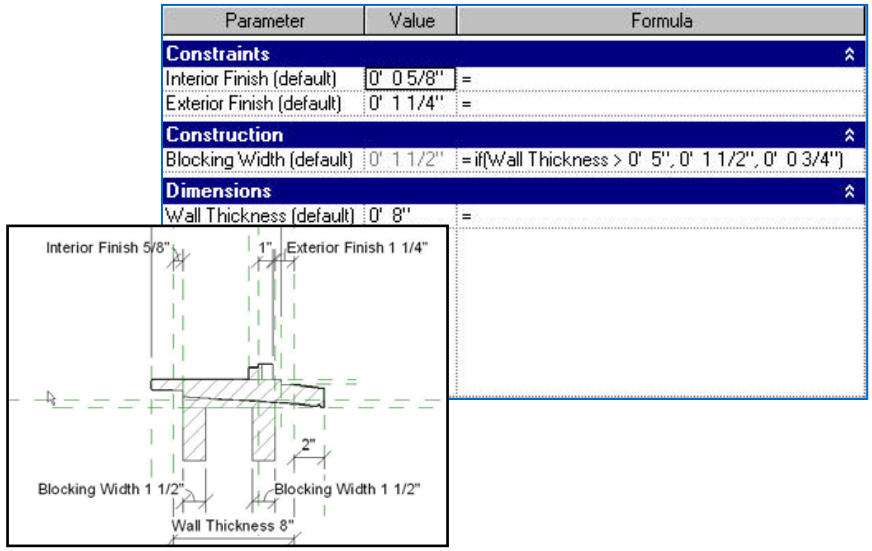
- Syntax
  - IF
    - =IF (Test Condition, Result if True, Result if False)

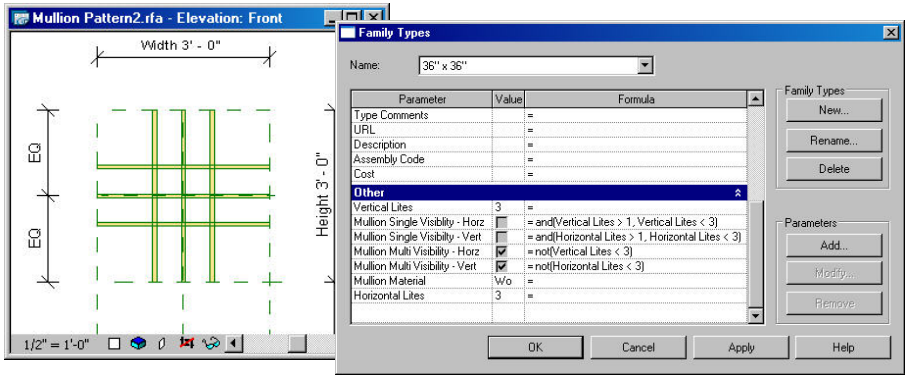
    IF (Length < 35' , 2'-6", 3'-0")
  - NOT
    - Inverts tests
    - For simple inversion of YES/NO parameters, no IF is required

    IF (NOT (A > B) , 8 , 3)
  - AND, OR
    - Test multiple conditions

    IF (AND (x = 1 , y = 2), 8 , 3)
  - Nesting
    - Alternatives if first test is false

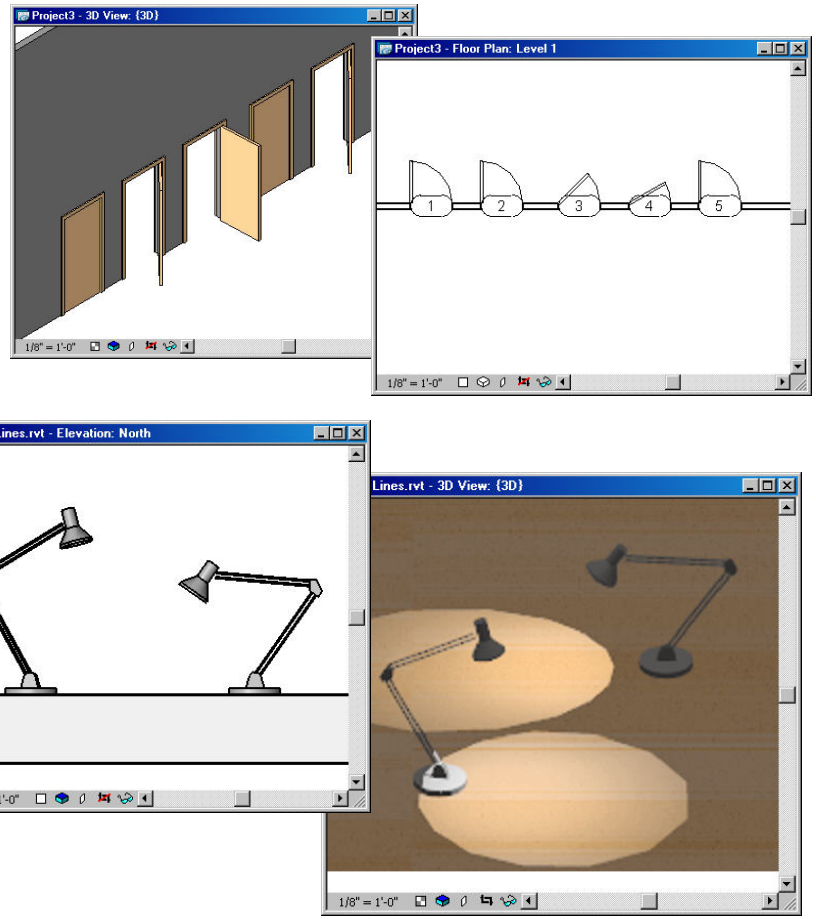
    IF (Length < 35' , 2' 6" , IF (Length < 45' , 3' , IF (Length < 55' , 5' , 8')))





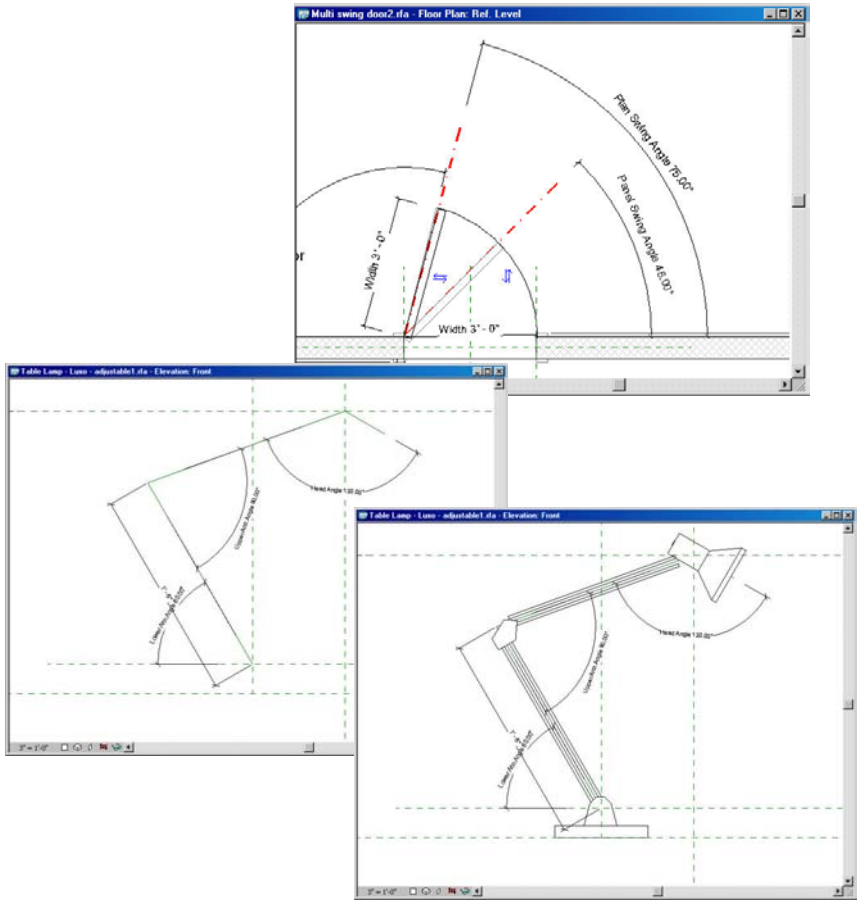
Good References: Reference Lines

- New in 7.0
- Control position and orientation without nesting
- Create self contained reference systems
- Move many elements with a single control
- Define movement linkages between elements



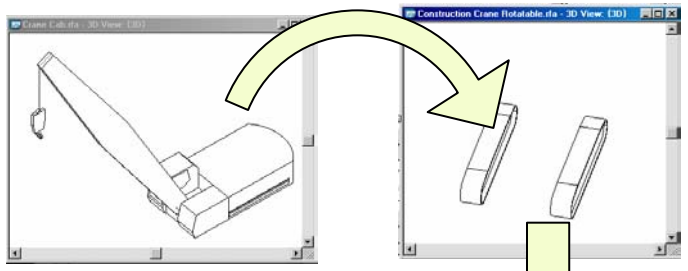
Good References: Using Reference Lines

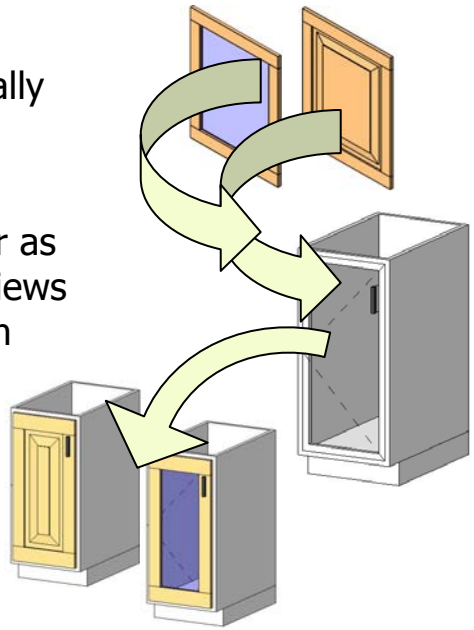
- Create
  - Select a work plane for the Ref. Line
    - Movement and rotation will occur in this plane
    - Work plane can be from another Ref. Line
  - Use Reference Line tool
  - Draw like ordinary model lines (straight lines only)
  - Can be dimensioned and constrained
- Using Ref. Lines for Work Planes
  - Use Plane tool on Options Bar
  - Select "Pick a Plane"
  - TAB when over the Ref. Line to switch between the 2 possible planes.

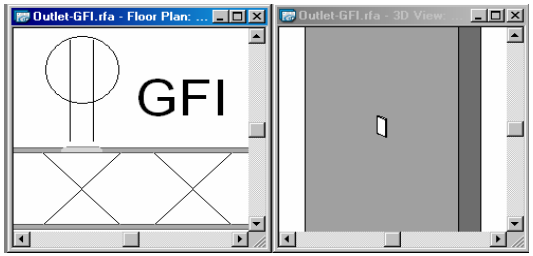


The Nesting Instinct

- Build families from subcomponents
  - Assemble Model elements
    - Reliable control of complex parts
    - Memory savings
    - Reuse your work
  - Swap subcomponents parametrically
  - Add Annotation elements
    - Put text in model families
    - Create families that appear as model elements in some views and annotation elements in others

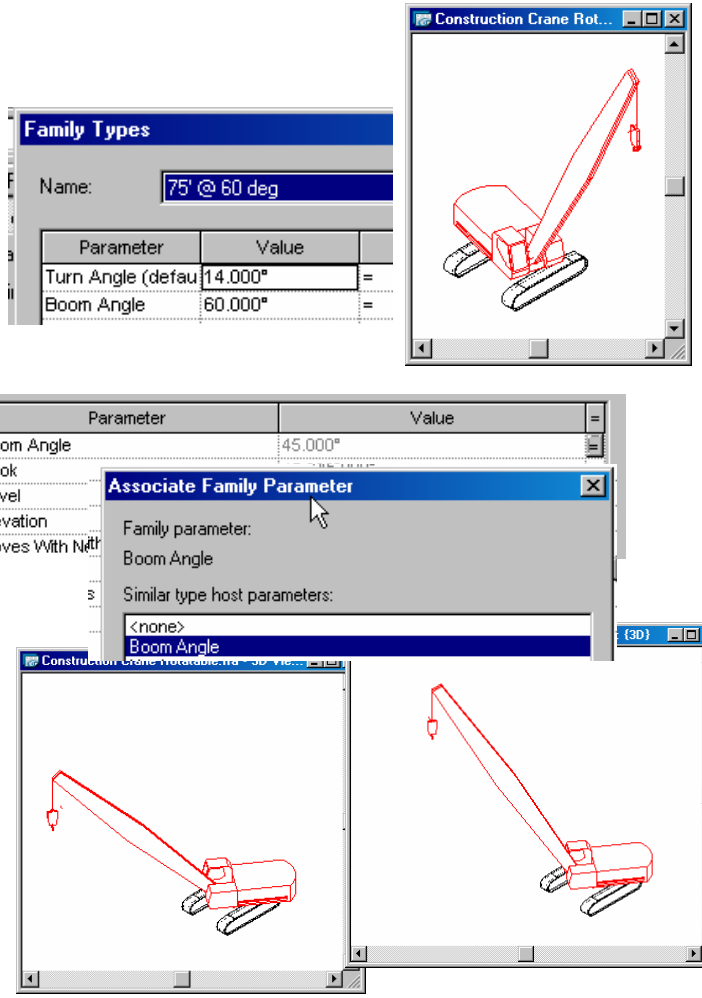






Controlling the Nest

- Move nested families parametrically
  - Align and lock to other elements
  - Use labeled dimensions directly
- Drive properties with parameters of host family
  - Create driving parameters in host FIRST
  - In nested family's properties, select = button and select from host family's parameters
  - Valid for type or instance parameters
- Control visibility of instances as a whole
  - By view
  - By level of detail

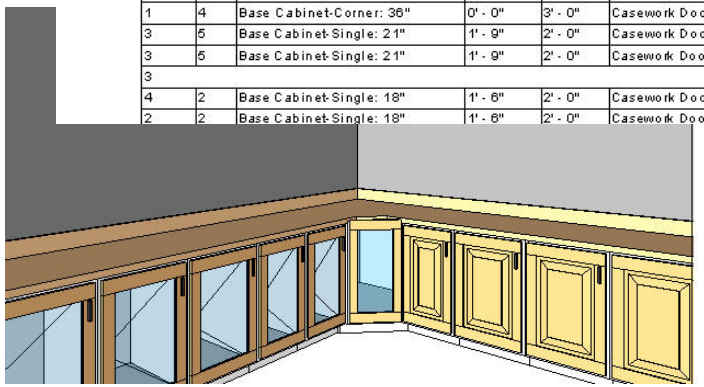


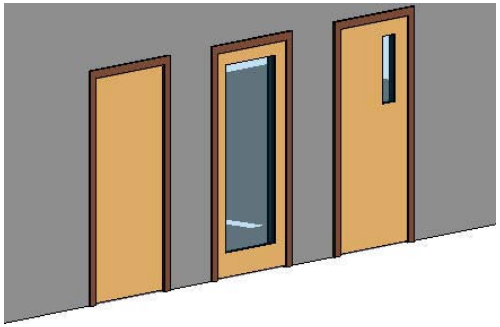
Controlling the Nest: Family Type Parameters

- New in 7.0
- Create Families with interchangeable parts
  - Parametric control of the family and type of nested family instances
  - Swap any nested family instance with an instance of any other Family/Type in the same category
  - Show the nested Family and Type in schedules
  - Control on either a Type or Instance basis

Casework Schedule

Count	Type Mark	Family and Type	Width	Depth	Door Style
1					
4	2	Base Cabinet-Single: 18"	1' - 6"	2' - 0"	Casework Door - Glass Panel
2	2	Base Cabinet-Single: 18"	1' - 6"	2' - 0"	Casework Door - Raised Panel
1	4	Base Cabinet-Corner: 36"	0' - 0"	3' - 0"	Casework Door - Glass Panel
3	5	Base Cabinet-Single: 21"	1' - 9"	2' - 0"	Casework Door - Glass Panel
3	5	Base Cabinet-Single: 21"	1' - 9"	2' - 0"	Casework Door - Raised Panel
3					
4	2	Base Cabinet-Single: 18"	1' - 6"	2' - 0"	Casework Door - Glass Panel
2	2	Base Cabinet-Single: 18"	1' - 6"	2' - 0"	Casework Door - Raised Panel





Using Family Type Parameters

- Creating the Parameter First

When you are building a template family

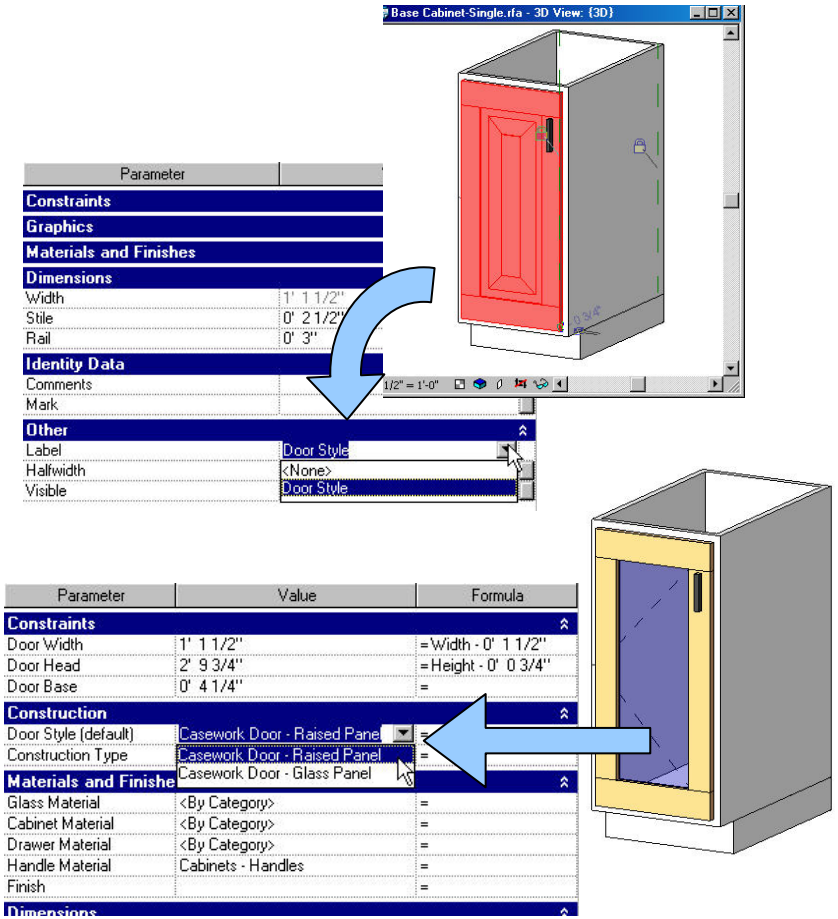
- In the family types dialog create a new parameter of type <family type>
- Select the category to control
- Load a family and place an instance

- Create the Parameter After Placing Instances

Lets you decide later

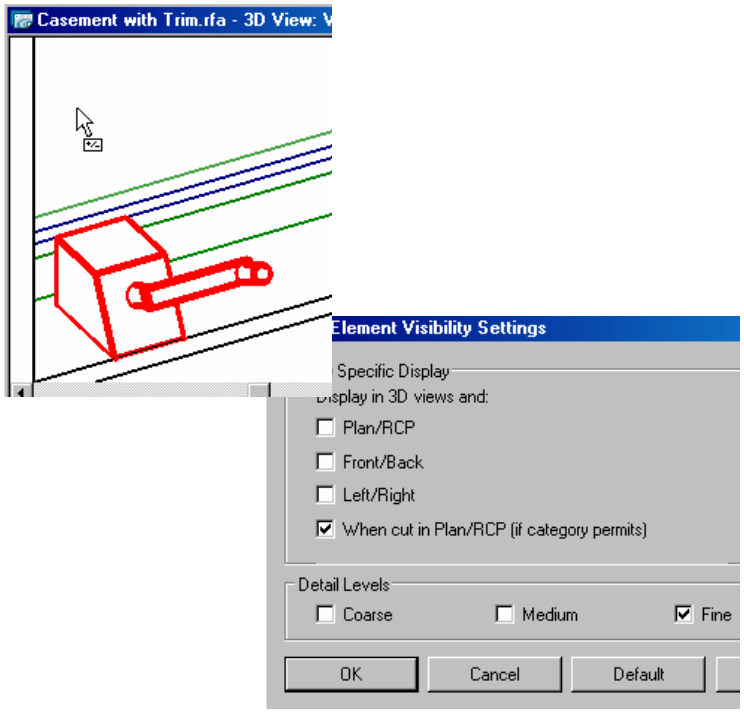
- Select a nested family instance
- On the Options Bar, select Label/<add parameter>
- Enter a name for the parameter.

- Show on Schedules
  - Use a shared parameter



Leverage Level of Detail

- Get more drawing information out of your families
  - Change from diagrammatic to specific representations
  - Add elements at increased levels of detail
  - Prebuild graphics required for detailing
- Increase performance
  - Reduce visual complexity in coarse views to make work faster
  - Simplify 3d elements to save on view generation time
  - Speed detail generation



New Features in 7.0

- Family Type parameters
- Reference Lines
- Use Length instance parameters in formulas
- RPC 3 Cars and Office Clutter available

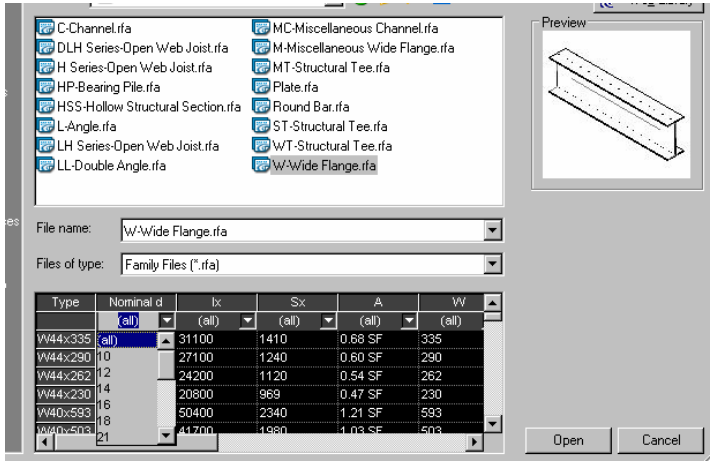
Questions?

- Thank you for your participation
- Power Point and Handout will be available for download
- Please fill out the survey.


AU
AUTODESK
UNIVERSITY
2004

Shorten the Family Tree

- Use Type Catalogs when there are many predefined types
  - Shortens the Project Browser tree: only explicitly loaded types appear
  - When loading families, the filter function of type catalogs gets users to the desired type faster
  - Manage types in an external editing environment without opening each type in Revit, or even opening Revit
- Drawbacks
  - A catalog file must be maintained for each family



Making Arrays Work for You

- Control arrays parametrically
  - Select array and then the dimension, add a label
  - Choose Drag to end while creating array to control overall length
  - Choose Drag to 2nd to control spacing
  - Select Append to End to have an end dimensioned array grow past the dimensioned element as number increases
  - To control the size of array elements use nested families and drive parameters from the host
  - Control array number as a function of an instance parameter. As length is dragged out, add more elements