Advanced Techniques for Managing Building Data in Autodesk® Revit®

Mario Guttmann – CASE

AB1796 The collection and management of building data, including the requirements for rooms, equipment, and other needs, as it evolves from early design through construction documentation and as a basis for facility management, can be enhanced through advanced techniques in Autodesk Revit software. This includes linking to an external data source, automating the creation of areas and rooms, creating room data sheets, and other graphic and non-graphic processes that complement BIM. This class covers basic database theory, the structure of objects and their relation to data management, and Revit techniques for implementing this general theory in actual projects. We will also discuss third-party software and customizations, and how they are used to further advance these processes. The class goes beyond learning Revit commands and looks at the underlying information requirements of emerging integration in project delivery.

Learning Objectives

At the end of this class, you will be able to:

- Understand how building data is represented in objects and databases.
- Evaluate computer system architectures and software options.
- Use Revit to study conceptual design and building massing.
- Create room data sheets in Revit and in an external database.

About the Speaker

Mario Guttmann, AIA, LEED AP, is a Senior Design Technology Specialist with CASE, a consultancy that provides technology-driven process innovation to the AECO industry. In this role he is engaged in research and development of work processes and software, and their application to design practice.

Mario holds degrees in mathematics and architecture, and is a licensed architect who has practiced for over 30 years. He has also worked in software development, computer-aided facility management, and construction. He is an active participant in industry standards organizations, and a frequent speaker in support of Building Information Modeling and Integrated Project Delivery.

His recent work has focused on strategies for engaging users, overcoming their objections, and providing solutions to the difficulties they encounter. He is an avid computer programmer, using rapid prototyping strategies to promote best practice and advanced design capability, and to provide direct value to projects.

M.Guttmann@CASE-inc.com
## Advanced Techniques for Managing Building Data in Autodesk® Revit®

### Contents

I. Introduction to Building Data
   - The Problem .................................................. 3
   - Information Flow ........................................... 3

II. General Principles of Objects and Databases
   - Objects and How Revit Uses Them .......................... 4
   - Basics of Relational Databases ............................... 5
   - Revit Objects and Relational Data .......................... 7

III. System Architecture and Software Options
   - System Architecture .......................................... 8
   - Software Solutions ........................................... 10

IV. Conceptual Design and Building Massing
   - Formalizing the Space Program .............................. 12
   - Linking Revit to a Database .................................. 13
   - Aggregate Planning Areas ................................... 14
   - Creating Rooms .............................................. 15
   - Supporting Construction Documentation .................... 17

V. Room Data Sheets
   - Numbering Rooms ............................................ 18
   - Room Data Sheets Overview ................................ 18
   - Room Data Sheet in Revit .................................... 19
   - Room Data Sheet in Access .................................. 19

VI. Supplemental Topics
   - Parent-Child Family Relationships – Site FAR Example .... 20
   - Related Database Tables in Revit Color Fill Plans ......... 21

VII. Example Project .................................................. 23
I. Introduction to Building Data

This class is about managing Building Data. This is the information about rooms, equipment, and so on that describes what goes into a building project. (It is differentiated from Project Data, information about the team, budget, schedule, etc., which we will not be talking about.)

A. The Problem

The design professions are experiencing a huge influx of data, and we are being asked to process it, share it and deliver it to others. It comes to us from a variety of sources: the architectural space program; user-group interviews; our own experience; and many others. We work with it in BIM and as data to produce drawings and other kinds of reports. Increasingly, we are also being asked to deliver the data itself, in both its BIM and database formats.

Between the source data and the deliverables, we work within a Database System, which includes both a BIM model grouping as well as a database of some kind. The need for a database, and the balance between it and the BIM component, are somewhat debatable: there are advocates for putting all of the data in the BIM model, while others try to put as much data as possible in the database. For the purposes of this discussion we will assume some kind of balance but stipulate that there must be a significant database component. How we manage the division of data between the two containers, and how we keep them synchronized and consistent, is the major focus of this class.

B. Information Flow

When we receive the owner’s space requirements, for example, they are often in paper or .PDF-file documents that are formatted for readability. In order to work with this data we need to transform it into a structured database, and then use it to develop rooms in the Revit model. Ultimately we want to be able to create reports that compare the spaces provided in the building with the original requirements. There are many information flows like this, for furniture and equipment, engineering requirements, and so on, all of which require special reports and other kinds of data deliverables.

Typically, project teams purchase software for this purpose. Unfortunately, using it is often problematic and the response is to provide additional training in the use of the software. While important, this overlooks a more fundamental issue: that the team does not understand its data. As a result, users need to memorize complex command sequences, and are unable to address unique requirements.

The core assertion of this class is that we are better served by first examining our data in terms of basic data principles. This understanding provides a conceptual framework that can be used to easily implement the software and improvise customized processes that it doesn’t provide “out-of-the-box”.

First, we want to understand our data, and define objectives for using it.
Then, we can implement software to achieve those objectives.
II. General Principles of Objects and Databases

This section describes some core principles of objects and databases, and how they are linked to one another.

A. Objects and How Revit Uses Them

A Model, as used in the context of Building Information Model, is really an Object Model. It utilizes some theory, developed in computer science around the 1950’s, for Object-oriented Programming. This includes important and very powerful concepts that, while fairly technical, really just codify ideas that have been understood intuitively within the design professions for a long time. To begin, we already understand what an Object is – a door, or a chair, for example; the theory simply formalizes this by being precise about Methods (what an object does) and Properties (the data attached to it.)

1. Object Inheritance

Of particular concern to the topic of data in Revit is the concept of Object Inheritance. Computer programmers don’t just create objects in an ad hoc manner; they typically start with a base object and then derive other objects from it. This is analogous to saying that there is some “grandfather” object, and that the other objects are its children, grandchildren, and so on. This is important because each subsequent generation inherits all of the properties of its parent, and then adds properties of its own.

In Revit, we usually call objects Families, and their properties Parameters, but that is mostly semantics. The significance of inheritance is that it explains how a parameter behaves in terms of its position in the inheritance tree. Some properties, such as the Element ID, are very basic and every Revit entity has one. Others, such as their hosting type and OmniClass value, are picked up as more specific objects are derived. Eventually we get to the family Types and finally to the family Instances, where we track the most detailed information such as model-type, size, and material.

2. Issues with Revit

There are some aspects of Revit that make this a little confusing. Shared parameters and the distinction of Project from Family parameters, while important, are not significant to the current discussion. What is worth noting is that the term “Family”, as used in the Revit term “Family Parameter”, is something of a misnomer in terms of objects. While we can simply and clearly talk about “Type” and “Instance” parameters, using the same logic at the family level to refer to “family” parameters would be confusing, so we tend to use a term like “built-in” or “system” parameters.

The user-interface is also somewhat misleading in that it allows the editing of type values from the modeling environment, in which really only the instances values can be changed, and the editing of instance values in the family editor, where they are really default values, since editing an instance at the type level doesn’t make sense.
3. Parameter Inheritance Levels

In project work, three levels of object inheritance are the most important:

**Built-in:** A limited number of very basic values that cannot be configured (even through the Revit API) and are edited only within the Family Editor in a section named “Family Parameters”. (This adds to the naming confusion; “family” is the right term from a computer programming perspective, but conflicts with the use of the term in other parts of Revit.) From a data perspective, the OmniClass Number is significant, since its application at this level means that it is the same for all types. This suggests against using very general-purpose families that might represent different kinds of building components.

**Type:** Parameters that apply to a Class of objects, such as a particular model of a manufactured item. Defining logical types is an important aspect of structured thinking about data.

**Instance:** Parameters that apply to single placements of objects. These are used where it would be impractical to define the many types that would be necessary otherwise, such as a specific finish material.

The proper balance between defining more types or having more instance variation is very important, and sometimes difficult to resolve. Nevertheless, it is important to understand that the data expressed is essentially equivalent, and either approach can be used to achieve any given result, such as a schedule. Further, Schedule Keys can be used to make instance parameters behave more like type parameters.

B. Basics of Relational Databases

There are other kinds of databases but we will be talking about the most common ones called *Relational Databases*. They are beautifully simple and surprisingly powerful. For some reason the BIM user community has neglected learning about them, which is a huge shortcoming.

1. Tables and Relationships

A database consists of *Tables* and *Relationships* between them. This simple construction can be used to represent a wide range of data, in a way that is very easy and efficient, but imposes restrictions on how data can be stored.

Each table is composed of *Rows* and *Columns* (which are sometimes called *records* and *fields.*) For example: The Room table contains a record for each room, with fields for Type, Area, and so on.

2. Primary and Foreign Keys

Every table has a *Primary Key* field, which is often called the *identifier*, and often has a name that includes “ID”. Its value must be unique, and must not be missing, so that it can be used to uniquely identify each row. For example: The RoomID field could be the key field of the Rooms table. (Tables can also have multi-part keys.)

Relationships between tables are established by a *Foreign Key*, i.e. a field in one table that is the key value in another table. Usually the foreign key field is allowed to be null, i.e. blank, which means that there is no association for that record.
Every table must have a Primary Key. Relationships are defined by Foreign Keys.

Generally these relationships are One-To-Many, which means that the “one” side must be unique but the “many” side may have many rows with the same value. For example: one FloorID value in the Floors table can be assigned to many room values in the Rooms table, but not vice-versa.

3. Referential Integrity, Normalization and Flattening Data

A database enforces Referential Integrity rules that ensure that you can’t corrupt the foreign keys. For example: a value cannot be added to the FloorID field of the Rooms table if a corresponding key value doesn’t already exist in the Floors table.

The database tables and their fields, and the relationships between them, should be well thought out so that information is stored once, and only once. This is done through a process called Normalization in which tables with redundant data are separated into related tables with unique values.

Generally it is better when data is normalized because it is easier to maintain and easier to understand. Unfortunately, there are many situations, particularly when working with Revit, where it is necessary to de-normalize the data, which is often called Flattening it. This is the opposite of normalizing: separate related tables are combined into a single table with some redundant data.

A table is **Not Normalized** if it has redundant data.

The data can be **Normalized** by using two tables.

**Normalized data is preferable but flattening data is often necessary with Revit.**
C. Revit Objects and Relational Data

Using these fundamental principles of objects and databases we can define what it means to *Link* them.

1. Linking Families to Tables

The link can be somewhat mysterious since there is no actual, tangible "link" item. Rather, the link is defined by a convention: the values of specific family parameters are identified as corresponding to data items in the database. In other words, if we took the link away it would have no effect on either the Revit model or the database; the link only exists insofar as some software recognizes the patterns in the data as constituting a link.

Generally, a class of family instances in Revit, such as a specific Revit *Category* like Furniture, corresponds to a table in the database. Likewise, the types of that family class may have a corresponding table in the database, often with a name like FurnitureTypes.

This is the first of many strategies that raise logical issues about redundant data. We are not going to solve these but it is important to acknowledge them and set expectations for how they will be managed. Ideally, well designed software will handle these issues but that is not always an adequate solution.

2. Linking Parameters to Fields

The parameters of the Revit families correspond to the fields of the database. One of these pairings must be between a *Key Parameter*, which can be any parameter designated for this purpose, and the *Key Field* of the table. This is the logical relationship that defines the link. Usually the parameters are something like “Room Number” or “Mark” and map to fields with the same name, but any choices are possible.

What is essential is that both the parameter and the field are not missing and have unique values. Revit can be a little sloppy about this; it will warn you about duplicate room numbers but it will allow them and they can be blank. Either of those events will destroy the validity of the link, at which point the data relationship is useless.

Additional parameters, *Dependent Parameters*, can be paired with other fields in the database on a one-to-one basis. There is no limit on the number of these and their values can be null.

3. Synchronization

Of course the logical pitfalls exposed with the tables manifest at the parameter level as well. Many times this means that there is data duplicated between the model and the database. Some process of *Synchronization* is essential to resolve this. These discrepancies are not necessarily errors; they represent the decisions and changes that are a part of the work process. Doing them in this structured fashion is how we are able to replace a lot of manual, error-prone, drudgery with elegant, accurate, automation.

Key schedules in Revit pose a particular challenge. They are, in effect, a kind of related table construction in Revit, which doesn’t support that level of database functionality in a true database sense. They can be very useful but coordinating their data with a true related table on the database side is complex.
III. System Architecture and Software Options

Using this theoretical framework of Objects Linked to Data-Tables, we can look at software strategies for doing the work. These can be categorized in terms of their System Architecture, which is a computer term for how the parts of a computer system fit together (not to be confused with “building architecture”, which is what we do in the AEC industry.)

A. System Architecture

These can be categorized by whether they are single or multiple-user, and the kind of multi-user technology they use.

1. Single-User:

The simplest approach is to have the data, including a database and other files such as images, on the same local computer where Revit is running. The Revit session includes an Add-In, which is a custom program running inside Revit. The add-in may connect directly to the data sources, or there may be an additional Windows application that functions as an intermediary.

While this system architecture does not support multiple users accessing the data, it is actually widely used to get work done in large organizations. It is a common misconception to think that collaboration requires simultaneous access to the raw data; in fact, it is often more effective to have a single user who maintains the data and produces reports that are shared with a larger group.

2. Multi-User – LAN / WAN:

Within an organization’s Firewall, which is the network environment controlled by that company, the data may be made accessible on the Local-Area Network (LAN) of a single office or a Wide-Area Network (WAN) that includes multiple offices.

Revit is able to use a shared Central Model and with building data it is used in the conventional way. However, it is worth noting that it is really the local copy of the model that is being synchronized with the data, so the various users are actually accessing local copies of the BIM portion of the database system.

Unlike the Revit model, the database and other files can be truly shared on a network server. The database may be actually a single-user type, which means that it can only be accessed by one user at a time, but this is really just a variation on the single-user system architecture.

A truly Multi-User Database, which is designed to manage multiple, concurrent, user access, typically uses a technology called Client-Server. This is a very mature technology that is quite robust and relatively fast but requires significant IT support.
3. Multi-User – Internet:

The most truly shared, and most forward-looking technology is what is commonly called The Internet, The Web, or The Cloud. The meaning of these terms is evolving rapidly and there is debate about their proper usage. From a technical standpoint there is some overlap with the LAN / WAN solutions but in the marketplace the Internet products tend to be hosted externally and support collaboration outside of an organization’s network and firewall.

Solutions of this kind typically include a remote Host, which is a computer complex that may be anywhere in the world. This host runs several software applications: a Web Service, which communicates with users; a Business Application, which is the program that does the work we are interested in; and, a local version of the database and other files that are utilized by the host application.

Historically, users communicated with this web host through a Web Browser, such as Microsoft Internet Explorer or Google Chrome, and this can be a good solution for working with building data. However, solutions are increasingly using a Rich Internet Application, which is similar except that the hosted application and the local Internet application are tightly integrated. This means that the program has the power of a local, Windows application, at the same time as it can do much of its work on the remote host.

This approach supports very wide collaboration since users typically only need a browser to access the project. Consultant engineers, the contractor and sub-contractors, the client and the community can all participate easily. Moreover, it is easier to maintain the software since it is all delivered from a single location.

However, this system architecture difficult to setup and maintain, and is more restricted in the kinds of data processing activities that the primary users can do.

All of the solutions have inherent integrity weaknesses. The multi-user options, and especially the Internet-based approach, compound these. There are no simple solutions for this problem; it is a requirement of any solution that, using good software design and proper user behavior, the various data repositories are kept synchronized.
B. Software Solutions

The purpose of this section is to contrast software options in terms of their system architecture. It is fairly difficult to get this information and to diagram it accurately. The vendors are more focused on the features of their products and the workflows that they support, apparently believing that users don’t need or want to know that much about what is going on in the background. While this assumption may be valid for the majority of end-users, this discussion is aimed at the more advanced users, technical support staff, and policy makers who need to care about the system architecture.

These applications are designed for very different purposes and have many complex features. In this discussion we are not evaluating those aspects, and potential users are encouraged to contact the developers to learn more about the products and services that are offered.

A selection of the many software solutions are described in the remainder of this section, ordered roughly from the simplest to the most collaborative.

1. Ideate – BIMLink

Fundamentally, BIMLink from Ideate is simply a tool for linking a Revit model to Microsoft Excel. As an inexpensive, and extremely easy to use, tool, it is seeing wide usage for many purposes. For users that want to use Excel (and this is very common) it provides an easy way to coordinate building data with Revit.

Excel is not really a true database, and BIMLink is not especially focused on supporting building data workflows, so this is a fairly limited solution. Nevertheless, it is easy to set up and keeps the team focused on their data, rather than engaging in a more complex software implementation.

http://www.ideatebimlink.com/

2. WhiteFeet Tools

Developed by a single programmer as a personal passion, The WhiteFeet Tools for Revit suite is not really a commercial application to same level as the other options. The suite also includes other commands, in addition to the database tool, which are used to illustrate the workflows in the next section of this document. (Full Disclosure: The creator of these tools is the author of this document.)

This solution is the simplest of the listed products that uses a true database. One option is to utilize Microsoft Access, which, in addition to being a good solution, will encourage wider understanding and use of databases. Alternatively, Microsoft SQL Server or the open-source MySQL databases can be used for a more enterprise-type solution.

http://www.whitefeettools.com/

3. Trelligence – Affinity

A product that is very appealing to architectural space programmers and early designers, Affinity from Trelligence is actually a stand-alone application with its own proprietary graphics system and database. It is focused on the architectural programming and early design stages so it is very useful during the initial building data workflows.

Affinity can link to various other BIM authoring products, including Revit, but its key strength is in its ability to standalone and remain independent of the subsequent BIM platform selections. Some projects use it during early design, and then switch to another product during the latter stages.

http://www.trelligence.com/
4. **CodeBook**

Probably the most mature of the options, *CodeBook* has achieved significant use in healthcare, particularly in the UK. Initially based on linking AutoCAD to Access, it now supports SQL Server with Revit and other BIM platforms. Projects can easily be scaled, from initial stand-alone exercises, to a multi-user production mode.

The basic design, using a client-server database in connection with a CAD/BIM graphics environment, is similar to the design of many products in use for Computer-aided Facility Management (CAFM). This is a well-established solution, with many production enhancements, especially in the management of furniture and equipment.

http://www.codebookinternational.com/

5. **Nosyko dRofus**

The *dRofus* product from Nosyko is a very contemporary solution, reflecting the current trend towards rich Internet applications, based on open, industry standards. Originally used primarily in Europe for healthcare, it is now being widely adopted in the US and for use with other market sectors.

The commitment to a true “Cloud” solution makes it slightly more formidable to launch a project, but results in a very robust collaboration environment. It includes features for tracking changes and accountability on large complex projects.

http://drofus.com/en/

6. **Onuma Planning System**

The *Onuma* organization is widely engaged with the AECO industry-wide discussion on open collaboration and has developed many products, often offered in connection with their other services. Their *Onuma Planning System* has been adopted particularly by public entities and university systems that require wide public involvement.

This product takes the “Cloud” philosophy, and the support of open standards, to an extreme. It has been used in a variety of public demonstrations of how large groups can collaborate publicly to tackle large-scale urban design problems.

http://www.onuma.com/
IV. Conceptual Design and Building Massing

Using the WhiteFeet software and Microsoft Access, we can demonstrate how a typical data framework is used in a Revit project.

A. Formalizing the Space Program
At the beginning of a project, designers and planners work with the client to establish architectural program requirements for rooms, equipment, and other needs. This Architectural Program Data is typically provided as text or in Excel and must be restructured so that it can be used with Revit.

1. Overview
The core process occurs in three steps:
- The data is entered into an Access Space Program table.
- The Space Program table is processed to build a Rooms table.
- The Rooms table is used to create rooms in Revit.

In addition, intermediate steps make use of the data during the development of the overall building form, and ultimately the data will be used to validate the building design against the original program.

This is the most important stage in working with building data. Unfortunately, most projects hurry past it because they are rushing to meet a preliminary deadline; and as a result, they are not able to manage the data accurately or validate the design against the program later.

2. Attributes
As we go through this process we want to be consistent with the kinds of attributes that we track with the spaces. A design team may elect to do this in various ways; however, the use of two attributes is recommended:
- **Organizational Use**: The activity, department, or cost centers that will use the space. For example: “Accounting” or “Sales”.
- **Room Type**: The characteristics of the room or the functional use of the spaces. For example: “Office” or “Laboratory”.

These attributes are each stored in their own separate tables, which have foreign-key relationships to the space program and room instance tables since their values are apply to many different kind of rooms.
3. The Space Program Table

The **Space Program** table lists the space requirements in the fields:

- **Space Program ID (key)**: A unique key value that will permanently identify the requirement.
- **Organization ID**: The foreign key to the **Organization** table.
- **Room Type ID**: The foreign key to the **Room Types** table.
- **Quantity**: The number of this organization-type combination required.
- **Area Each**: The area required for this organization-type combination.

Often the space program will not explicitly call for circulation and other kinds of common space, or they may be expressed as a factor of the specified spaces. In these cases additional program entities must be created to hold these values so that it will be possible to validate the areas later.

**B. Linking Revit to a Database**

At this point we begin to establish the relationship between the database and the model. In this demonstration we are using the WhiteFeet Tools command **Revit-Database Link**. It uses special tables in the database to define the **Schema**, which is a list of the various tables, key fields, dependent fields, and Revit parameters that define the link. We will do this for **Areas**, which are used for general planning purposes, and for **Rooms**. Although not included in this example, a similar process would be used with furniture, equipment, and other elements.

Once the link is established, the tool is used to compare the Revit families with their respective data tables. A synchronization process is used to insure that the values are the same in both contexts.
C. Aggregate Planning Areas

During the early stages of design, rooms are too granular to be used easily to develop a building form. Instead, we use composite Aggregate Planning Areas.

1. Planning Areas in the Database

The planning areas are grouped by:

- **Organization Category**: Grouping of values from the Organization table.
- **Room Type Category**: Grouping of values from the Room Types values.

Note that we first define the more granular attributes, and insure that they have good identifiers, and then “roll-up” these values into categories, rather than the other way around. It is always much easier to group things by a characteristic than it is to try to divide an undifferentiated collection into groups after the fact.

2. Working in Revit

Using the database link and the WhiteFeet Area & Mass Tools → Place Unplaced Areas, we import aggregate areas from the database and use them to create unplaced areas in Revit. Note that we include the category values and the required areas in the import process, using them to populate parameters of the area objects. This means that we can place the areas as squares, colored according to their function, and sized according to their required area. Sometimes the blocks are rectangular, based on a standard planning module. These can be manipulated to develop a building form.

However, in actual project work it is usually misleading to think that a building form can be generated from the space requirements alone. In our example we are going to assume that the designer has an architectural “Parti” for the building. It includes a site with some constraints, a “U” shaped building footprint, a typical floor dimension that is based on the width of a classroom and a corridor, and a standard floor-to-floor height.

*An architectural “parti” based on site constraints, planning modules, and floor heights.*
Using the parti as a base, we can outline major planning areas with *Revit Area Boundary Lines*, and drag the areas into place. Once they are moved, the area objects adjust their sizes to meet the boundary lines.

At this point we can begin to develop the building massing by converting the areas into *Revit Mass* or *Revit Generic Solid* forms. The category attributes are transferred to the form so the blocks can be colored, which is used to create a 3D diagram of the design. The WhiteFeet *Area & Mass Tools ➔ Create Solids From Areas* is used to do this.

Because of the way we constructed the database link we can synchronize the planning areas with the database and compare the actual areas with the design areas to validate the scheme.

Walls, floors and roofs are developed around the area-data-driven masses.

D. Creating Rooms

Based on the planning areas we can begin to develop the architecture, including creating walls, floors and roofs. These imply room spaces, but do not yet have Revit room objects in them.

1. Rooms in the Database

   Before creating the room objects, we develop a rooms list in the database. As we do this, we need to insure that our organization and type values are structured correctly. Typically we give them a key value that is a short, capitalized string, such as “01”, “02” … or “OFF-1”, “CORR”, etc. Then, we put the more descriptive name in a *Description* field in mixed case. That way we can use the key values in tags in Revit, and show the more descriptive name in Revit schedules and Access reports.

   The example uses a VBA program in the Access database to generate the room records from the space program table. Being able to do this kind of simple programming is an important skill for an office to have because it means that it can automate a tedious task and customize the way it is done. This is another critical step where it is really important to plan the various ID values and relationships. As a minimum, from the *Space Program* table, we want to populate the following fields in the *Rooms* table:

   - **Room ID (key):** A unique key value that will permanently identify the room.
   - **Space Program ID:** A permanent reference to the program item that the room came from.
   - **Organization ID:** The foreign key to the *Organization* table.
   - **Room Type ID:** The foreign key to the *Room Types* table.
   - **Area Required:** The required area, transferred from the program.
The Rooms table does not have a Quantity value like the Space Program table. Instead, multiple rooms are created, according to the quantity specified. This means that there are many more room records than program entries, and that the VBA program has some strategy for generating unique ID values, such as appending a ".01", ".02", … suffix to the program ID values.

Ideally, the Room ID value should not be the same as the room’s Room Number value, which will likely change as the planners make adjustments to the design. Unfortunately, owners often specify that they be the same, so it is important to maintain at least the Space Program ID value as a pointer back to the original program so that the history of where the room came from is preserved.

The related Organization and Room Types tables are the same tables that were used with the Space Program table. This discipline is important because it insures that the attributes of the rooms that we will be using to validate the design will align with the original program.

### Rooms in Revit

The rooms are created in Revit as unplaced rooms using the database link again. Using the WhiteFeet Room Tools → Place Unplaced Rooms command tool the rooms are arrayed as squares inside of Revit Room Separation Lines or temporary walls, sized according to their required area.

As with the areas, these rooms have brought their category, room type, and other attributes with them as parameter values. It is helpful if, during the database work, the rooms were classified by the floor they will be used on. That makes it easier to place them on the proper floor plan.

Finally, they are dragged to their final location in the plan.
As the rooms are moved inside of the architectural walls of the building they assume the size of the space provided. This means that their Area parameter reflects the actual size of the room.

Using the database link again they can be synchronized back to a field such as Area Actual, and compared to the Area Required. This can be used to develop a report showing the comparison of what was asked for to what is now in the design. Because of the internal consistency of the data, these reports can be grouped by the organization and room type attributes.

E. Supporting Construction Documentation

In this demonstration we are not developing the database further, but in actual projects it can be very useful to use similar processes manage other aspects of the project and documentation. These can include:

- **Room Requirements**: Utilities, finishes, and other detailed requirements. These may be defined initially in the room types so that they apply generally, but will often move to the rooms themselves as the project progresses and there are more exceptions than typical cases.

- **Doors and Hardware**: Although doors are scheduled in Revit it is often convenient to manage hardware and other detailed requirements as data.

- **Furniture and Equipment**: These may be managed as data initially and then used to develop furniture and equipment plans.
V. Room Data Sheets

The *Room Data Sheet* process involves working with client *User Groups*, which are made up of both experts and general users who know what is required in the building. This information must be gathered systematically and used to drive the design. In this example project the synchronization of this detail between the Revit model and the database is not developed, although such functionality is a major component of some of the commercial software options.

A. Numbering Rooms

Up to this point the rooms have been identified by a *Room ID* value that was assigned in the database. Once the rooms have been placed they are usually numbered in their *Room Number* parameter in a sequence that makes it easy to find them on a plan, such as sequentially along the corridors.

As noted earlier, it is preferable that the *Room ID* value and the *Room Number* value not be the same since the ID value is based on the program requirements and the number is based on its position on the floor. In addition, the rooms will likely have a *Room Signage Number* parameter that defines what text will be on the plaque placed by the door. Although it is tempting to try to make these numbers the same, actually doing so is very difficult, and keeping them separate will save work in the long run.

The actual method of numbering the rooms is not part of this discussion and many utilities are available for doing that. It is also a good idea to synchronize the room numbers back to the database so that the mapping of the numbers to the corresponding ID values can be included in reports.

B. Room Data Sheets Overview

There are two distinct strategies for the *Room Data Sheet* process:

- **Room Data Sheets in Revit**: The sheets are developed entirely within Revit.
- **Room Data Sheets in Access**: The sheets use images from Revit but are developed as Access reports.

Both methods are valid and can be used simultaneously. The process in Revit lends itself to quicker adjustments while working in Revit and may be most useful to the design team. The Access reports can include more kinds of data and have more formatting options, so they may be more useful in working with user groups.

Both methods involve creating a separate view for each room. This is a small plan or 3D view that is cropped to a fixed distance around the room. The WhiteFeet command *View Tools → Create Views From Rooms* is used to create these in a batch process. The views are named systematically with a variant of the room number to enable the use of additional automation in the processes that follow.
C. Room Data Sheet in Revit

Each of the rooms is tagged, in the small room view, with a special tag that includes labels for all of the parameters that are going to be displayed. This is just a standard Revit Room Tag with many labels, which means that it will need to be repositioned if the walls around the room change. The WhiteFeet command View Tools → Add Tags to Existing Rooms is used to do this in a batch process.

Each of these tagged views is then placed on its own sheet. The sheet uses a special Revit title block that is designed especially for this purpose, usually with a letter or tabloid paper size. The WhiteFeet command View Tools → Create Sheets From Views is used to do this in a batch process.

D. Room Data Sheet in Access

The views in Revit are exported to raster image files in a batch process using the WhiteFeet command View Tools → Export Images From Views. The images are named systematically so that they can be linked to the Access report based on a standard pattern. The images should be placed in a folder in the same location as the Access report.

A form in Access is used to input data, often live during user group meetings. A separate report is used to publish the same information in a read-only mode.

Creating the input form and the report in Access is a complex topic, outside of the scope of this discussion. They are based entirely on data from the Access tables so any data from Revit must be separately synchronized.

The exported images can be displayed in both the input form and the report. The syntax for the Access Control Source is of the form:

\[=\text{[CurrentProject].[Path]} \& \"\text{Images}\text{\slash}\text{Room}\" \& \text{[Number]} \& \"\text{-3DBC.jpg}\]\n
In Access, a form used to input room data, and a report used to create a booklet, have a similar format.
VI. Supplemental Topics

A. Parent-Child Family Relationships – Site FAR Example

In many cases, it is useful to define “ownership” relationships between different kinds of families. These occur when there is a logical one-to-many type relationship. They are commonly expressed as one family “owning” the other, or that one is the “parent” and the other is the “child”.

For example, in site planning it is often necessary to calculate a Floor Area Ratio (FAR) that expresses the ratio of the total floor area of all of the buildings on a parcel to the area of the parcel. This kind of calculation only makes sense if each building “belongs” to one, and only one, parcel.

\[ \text{FAR} = \frac{\sum \text{Areas of All Floors}}{\text{Area of Parcel}} \]

Although Revit includes a Property Line object this class has limited functionality so it is typical to use Area objects for this purpose. (Custom values for the Area Schemes and Area Types are limited; so arbitrary values are typically used and then parameters added to store any site characteristics.)

To represent buildings, mass objects can be used, either modeled in-place or as separate families. A system of Revit Levels is developed for the entire site. For each mass object, when the appropriate floors are activated, Revit Mass Floor objects are created, and the total area of all of these for that building is calculated automatically.

1. Establishing a Parent-Child Relationship

   The ownership, or parent-child relationship is represented by the child storing an identifier of the parent that it belongs to as a parameter. (Note that we do not do the opposite – have the parent store the ID values of its children – since we don’t know how many children there might be and it is hard to work with value lists of varying length.)

   In the example, each parcel (Revit Area object) has a parameter named AreaId. These are populated with unique values to identify the parcels. Each building (Mass family) includes a parameter named ParentAreaId establishing which parcel it belongs to.

   In this example we have used the WhiteFeet Parameter Tools → Parent-Child Relationship command to find which buildings are located on which parcels and assign the appropriate ID values.

2. Rolling-up a Value to the Parent

   A numerical value such as area is said to be rolled-up when the values of the children are summed and the total is stored with the parent.

   By default, Revit stores the total floor area of each building in a parameter called Gross Floor Area. In the example, a parameter called AreaSumOfChildFloorAreas has been added to the parcels. The total floor area of all of the buildings on a given parcel is assigned to this parameter. This is fairly tedious to do manually, so the WhiteFeet Tools command Parameter Tools → Roll-Up Value to Parent has been used to do the arithmetic and assign the values.
3. Calculating a Ratio

Although Revit can do calculations in schedules, these values are difficult to use with tags and color fill plans, so it is often convenient to pre-calculate a value and store it in a parameter. In the example we have the AreaSumOfChildFloorAreas values calculated in the previous step, and Revit automatically reports the area of the Revit Area object in its Area parameter. We need to take the ratio of these and store it another parameter called FloorAreaRatio. The WhiteFeet Parameter Tools → Math Calculation tool has been used to do the arithmetic and assign the values.

4. Creating a View Filter

In the example, floors have been added to represent the parcels in plan views since the areas are not visible. A Revit View Filter, utilizing the FloorAreaRatio parameter, is used to shade the parcels on a scale. All of these processes use standard Revit commands.

B. Related Database Tables in Revit Color Fill Plans

Because Revit tends to flatten data it can be difficult to display data that is based on relationships and aggregations. This example uses several tools to illustrate various techniques for getting around this.

The goal of the example is to create a Revit Color Fill Plan that shows the rooms colored by their Room Type Category value in the Access database with a legend that shows:

- Each category value.
- Its descriptive name.
- The total room area of that category on the current floor.

To do this, the example illustrates several techniques:

- Indirect values: Values that must be retrieved from a related table.
- Rolled-up values: Summing up a value across a selected grouping.
- Concatenation: Combining text into a single parameter value.

1. Importing Room Type Category Values to Rooms

In the database table Rooms, each room is assigned a Room Type. In a related Room Types table each such room type is associated with a Room Type Category. In order to work with this data in Revit we need to “flatten” it into a single parameter of the room called RoomTypeCategory. We also want to bring along a more descriptive title, from the Room Type Categories table, and assign it to a parameter called RoomTypeCategoryDescription. The Update Derived Values option of the WhiteFeet Revit Database Link tool is used to do this.
2. **Separating Rooms by Room Type Category and Floor**
   
   In order to perform aggregate functions on the rooms we need to be able to group them properly. We do this by creating a new parameter called `RoomTypeCategoryFloor` and using the WhiteFeet Parameter Tools → String Calculation and Concatenation command to combine the `RoomTypeCategoryId` value from the previous step with the Revit Level number. The resulting values associate each room with a distinct floor and room type category combination.

3. **Summing the Room Areas**
   
   Using the `RoomTypeCategoryFloor` parameter from the previous step to group the rooms, the total area is calculated and stored in the `RoomTypeCategoryFloorArea` parameter. Note that this value is the same for all rooms with the same category-floor combination. In the example, the WhiteFeet Parameter Tools → Math Calculation tool has been used to do the arithmetic and assign the values.

4. **Converting the Number to a String**
   
   In the following steps we will be creating a legend that incorporates the `RoomTypeCategoryFloorArea` parameter value calculated in the last step. In order to use it this way it needs to be converted to a string value and stored in the `RoomTypeCategoryFloorString` parameter. The WhiteFeet Parameter Tools → Math Calculation command tool can be used to do this.

5. **Creating a Title Parameter with a Concatenated String**
   
   In order to create the final legend, all of the assembled data must be combined into a single parameter. In this case it combines three parameter values and three literal strings:
   
   - `RoomTypeCategoryId`.
   - Literal “–”.
   - `RoomTypeCategoryDescription`.
   - Literal “( “.
   - `RoomTypeCategoryFloorString`.
   - Literal “ SF )”.

   The combined string has the form: **CIRC – Circulation ( 6671 SF )**
   
   Its value is stored in the parameter `RoomTypeCategoryTitle`. As an illustration, the Update Derived Values option of the WhiteFeet Revit Database Link tool is used to do this; however, the concatenation tool could be used in the same way.

6. **Creating a Color Fill Legend with a Title Parameter**
   
   Now that we have the values in the `RoomTypeCategoryTitle` parameter, conventional Revit commands are used to create the Color Fill Plan and Color Fill Legend.

The final color fill plan based on the `RoomTypeCategoryTitle` parameter.
VII. Example Project

The images used in this presentation have been based on sample data, which is easy to work with and convenient for illustrating techniques, but is not truly realistic in terms of what occurs in actual projects. The images on this page, which are from a real project provided courtesy of Catherine Chan at HDR, illustrate what building data looks like when it is scaled up to hundreds of records. They also illustrate how flexible Revit and Access can be in creating special-purpose forms and reports for inputting and distributing data.