# IFC on Strange Matter

Strange Matter was born out of the needs expressed by the needs of both the buildingSMART community for future versions of IFC but also the large community around the build environment that is looking for a modern flexible way to connect data.

Strange Matter is an open protocol for working with distributed, heterogeneous data used before, during, and after all phases of design, construction and operations of the of the built environment.

Strange matter is format, vendor, and tool agnostic.

It is a way for people, processes, and tools with different requirements working together on design and construction projects to collaborate on data that has distributed ownership, comes from different sources, and that is continuously changing.

Strange Matter does this by providing a universal abstract concept of entity. That is the thing that people care about (whether it is a particular building, floor, facade, column, roadway, pile, rail bed, asset, or whatever) and for which more or less data may be available to different stakeholders over different periods of time, authored in different pieces of software.

Actual data is organized in components and relationships. Components are JSON headers that refer to data payloads, which can be in any format a user or tool generates. Relationships are defined in the same way as components, just without payloads, and can describe any kind of semantic relation between two components. A relationship between a component and entity is done by sharing a relationship with an Entity ID component.

# Strange Matter Design Criteria

Strange Matter was born from the key requirements of the industry. Chief among them is multiformat and multi standard support. It became clear that to reach this goal the protocol use to organize data had to be standard agnostic.

Beyond that the actual data representation was informed by broad community input most of which is captured here.

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Type** | **Design Solution** |
| Data is assembled across many sources. It changes over time, has many versions and states. A single thing might be simultaneously in design, construction and operations all at the same time. | Functionality | Entity Component Model where the Components are Packets of Data. This is augmented by relationships that are also components. This allows data to be continuously added and related together while having no impact on the existing data |
| To describe the built world we need GIS, BIM, text, requirements, issues, inspections, lidar data, point cloud data, products, carbon, energy just to name a few. No format, standard etc. can ever describe all this data so we need to find a way that can connect standard, nonstandard and even proprietary data without taking ownership of its description | Functionality | Strange matter is format, vendor, and tool agnostic. |
| Machine to Machine Readable | Functionality | There have been attempts to package heterogeneous data in formats previously, but the data was only labeled or classified and didn’t allow machine to machine communication. To solve this Strange Matter has proposed robust methods of data description and self-description of payloads |
| Heiarchy is in the eye of the beholder | Functionality |  |
| The data we use to digitally describe the built world relies heavily on relationships. This is often to the service of the format more than it is to the users of the data. |  | Relationships are just as easy to create and compose as the data itself. This is a key requirement to incorporate automation, workflow and future machine learning capacities. |
| Data needs to work in many platforms, offline, online, archive, |  | Strange Matter is rather simple in design but meant to work as a complete protocol as its indented to be whole, no matter the system that is used. It’s well suited to be stored, accessed and created in many technologies. It’s simple enough that it can work as a file on disk, but it certainly can be use in SQL, NO-SQL, Graph and other formats. The emergence of Columnar “File as Database” formats like [Apache Parquet](https://parquet.apache.org/) or [LanceDB](https://lancedb.github.io/lance/) that are becoming standard in the data science and ML world are very intriguing as they marry very well the component based approach. This opens the door for native ML and automation capabilities directly on the data. |
| Technology needs to enable solutions independent of contracts, governance or delivery models. |  | One of the many challenges in the built environment is the variety of ways that the facilities / infrastructure are organized and managed. Strange Matter looks to provide much needed technological flexibility to work in these many different models and to enable the benefits of digitization while doing so. |
| Flexibility in Adoption | Robustness | Moving from a file-based world to an all web world should not be a requirement for adoption and thus the system is flexible. In reality there is a strong interest in a middle ground that mirrors other distributed design and engineering driven industries like software development and Film and VFX, where file-based systems are used behind change and version management tools like Git and Github. |
| Embrace Web Technologies but don't lose data in doing so | Robustness | Strange Matter looks to use the web as much as possible to enable richer developer, user and data experiences but also acknowledges that while the web enables rich experiences it is not permanent. To allow for these two truths it looks reference the web when available but allow provide mechanisms to ensure that data collected in Strange Matter does not loose permanence if its web sources disappear. |
| Data history must be better supported and ownership clearer. | Provenance and History | Strange Matter is design to support this by allowing 2 modes of working. First is the simplest where in situations where no data change management system is in place the data is always immutable. This is possible because Entity, and Component UUIDS are stable for the life of objects. Only the Version UUID is to ever be changed. This can be used in conjunction with semantic versioning to have a copy of every single version of the data. In situations where a Change management system in place its possible for minor versions of the data to be overwritten but any published data should never be overwritten. This has a 1 to 1 parallel with how software code is used where WIP data can be updated but once it packaged it must be stored as a separate artifact. Thus, published data should never be mutated and stay immutable the same should be said where a specific component is authored and augmented by another application. The original data should never be mutated. |
| Data Ownership and Design Transfer. | Functionality | One of the most complex problems in this space is ownership of Geometric and non-Geometric Data by Data Creating Software. The solution to this problem is to move from an editing model to an additive model. Where Components that are authored by one tool cannot be overwritten by another tool. Instead, they must reference the components and make the linkage to the existing Component and Entities and create a new version of the components. This approach has many benefits. It allows for Complex Geometric Data to be used in Simpler forms when the goal is not to exchange ownership but where new components based on the existing data is all that is required. Also, it allows for the “gold version” of complex geometry to be maintained even when its necessary to convert geometry to an applications internal data model. This is possible because components can have more than one geometric representation that can be on the same entity linked by a derives relationship. |
| Share as You need (half) |  |  |
| definition is not instance data |  |  |
| Git |  |  |
| Data is independent of its container |  |  |
| Benefit for all Creators, Adders/Maintainers |  |  |
| Out of Order and Additive |  |  |
| Protocol not an API… |  |  |
| Fine Control on Security/ Access |  |  |
| Strange Matter has one component (currently) |  |  |
| Classsifcation is for funtion/ Component Type is for content |  |  |
|  |  |  |

# IFC on Strange Matter Design Criteria

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Type** | **Design Solution** |
| Simplify |  |  |
| location and positioning |  |  |
| Visuals |  |  |
| Explicit Units |  |  |
| Bools to Bools |  |  |
| Developer Experience |  |  |
| Class Structure is for Maintainers |  |  |

# What is Strange Matter and How does it work?

The main concepts behind Strange Matter come from Entity Component Systems. ECS. Instead of Inheritance-based objects and data it follows a Composition model. Here is a good technical background. [Leatherbee ECS](https://leatherbee.org/index.php/2019/09/12/ecs-1-inheritance-vs-composition-and-ecs-background/)

It should be said from the outset that ECS learns from but given the need to allow for a distributed approach it diverges in some key areas so while ECS is the president Strange Matter follows its own rules and so is best described as a means to compose data .

The graphic below illustrates the core ideas. A single entity might be fully described using several different types of data, from different standards and systems. Hence an object-based definition from classical programming is not well suited.

A diagram of components with text

Description automatically generated