Sparx Enterprise Architect Design Patterns

Matthew Lawler

Lawlermj1@gmail.com

Introduction

Licence

This document is released under the Creative Commons Zero licence or CC0.

Warranty

The author does not make any warranty, express or implied, that any statements in this document are free of error, or are consistent with a particular standard of merchantability, or they will meet the requirements for any particular application or environment. They should not be relied on for solving a problem whose incorrect solution could result in injury or loss of property. If you do use this material in such a manner, it is at your own risk. The author disclaims all liability for direct or consequential damage resulting from its use.

This is not a comprehensive document, as design patterns used will make clear. As each Sparx EA environment is different, so these choices will also be different.

Purpose

This document describes the design patterns used in Sparx Enterprise Architect (EA). Sparx EA can define, manage, and model metadata.

Audience

The primary audience for this document are designers, who use EA to maintain metadata.

Assumptions

It is assumed that the reader understands metadata, database design, data modelling, SQL, etc.

Approach

This document will only analyse the EA database design layer because this is most relevant for the task of metadata automation, especially data integration. It will describe good and bad design patterns that are used in EA to define types of design objects. The bad patterns can also be called anti-patterns, as they impact productivity by increasing complexity to the point of confusion These design patterns will help in showing how to use EA. It uses an example to show Sparx can be used.

Annoyances

EA is a commonly used tool for metadata modelling, which is why a document like this is needed.

EA is designed following OOP principles. EA relies on a graph database for its flexibility. But the downside of flexibility is complexity. The example uses a modified ISO 11179 standard, as well as entity relationship modelling. The combination of all these standards produces name collisions, which is the chief annoyance of EA. Critical terms such as Attribute, Class, Element, Object and Property are reused and redefined within the layers of the tool. Then there is the OOP propensity of renaming, which adds to the annoyance. This leads directly to comprehension issues, which means that it can be very difficult to reason about the designed models. EA controls this issue mainly by recommending that the users only use the EA GUI.

However, if the goal is to automate the design process, then a deep understanding of how the EA DB works is required. This is certainly the case when integrating with other metadata tools. In that sense, this document can be a dissection or X-ray view of EA, which will turn EA into a white box, so that automation becomes possible.

By

This was written by Matthew Lawler. It is based on data profiling and code reading. The documented patterns most closely resemble RDF or OWL design patterns, as Sparx is a hybrid of relational and RDF concepts. My goal was to make everything as simple as possible, but not simpler.

Acronyms

This is a list of acronyms used in the document. ML refers to Matthew Lawler.

0	Acronym	Expansion	AKA	By
1	AKA	Also Known As		English
2	ANSI	American National Standards Institute		ANSI
3	CROP	Column Reification Object Pattern		ML
4	DAG	Directed Acyclic Graph		Maths
5	DAMA	Data Management Association		DAMA
6	DB	Database		ANSI
7	DROP	DAG Reification Object Pattern		ML
8	EA	Enterprise Architect		Sparx
9	EAOM	Enterprise Architect Object Model		Sparx
10	FK	Foreign Key		ANSI
11	FQ	Fully Qualified		Sparx
12	GROP	Graph Reification Object Pattern		ML
13	GUI	Graphical User Interface		OMG
14	GUID	Globally Unique Identifier	UUID	Microsoft
15	IP	Intellectual Property		English
16	ISO	International Organization for Standardization		ISO
17	KV	(Key, Value)		ISO
18	MDG	Model Driven Generation		Sparx
19	OMG	Object Management Group		OMG
20	OOP	Object Oriented Programming		OMG
21	OWL	Web Ontology Language		W3C
22	PK	Primary Key		ANSI
23	PROP	Parent Reification Object Pattern		ML
24	PV	(Property, Value)		ISO
25	RDBMS	Relational Database Management System		IBM
26	RDF	Resource Description Framework		W3C
27	RO	Read Only		OMG
28	ROP	Reification Object Pattern		ML
29	RW	Read Write		OMG
30	SA	Subject Area		DAMA
31	SQL	Structured Query Language		ANSI
32	SSMS	SQL Server Management Studio		Microsoft
33	TA	t_attribute		Sparx
34	TAT	t_attributetag		Sparx
35	TC	t_connector		Sparx
36	TOB	t_object		Sparx
37	TOBP	t_objectproperties		Sparx
38	TP	t_package		Sparx
39	TROP	Type Reification Object Pattern		ML
40	UI	User Interface		OMG
41	UUID	Universally Unique Identifier	GUID	ISO

0	Acronym	Expansion	AKA	By
42	W3C	World Wide Web Consortium		W3C
43	XMI	XML Metadata Interchange		OMG
44	XML	Extensible Markup Language		W3C

Definitions

As a general comment, this document describes the database of a tool which manages metadata data. This leads immediately into a language confusion trap, and the classic 'name collision' problem. Many standard data modelling terms are redefined within the tool, which is extremely confusing to anybody with some data knowledge. This problem is not unique to this tool, as it re-emerges in other contexts related to ISO 11179 implementations. The best way to resolve these ambiguities is to look at examples, which clarify the meaning behind overly abstract terms. These redefined terms are captured and defined below. Some of this confusion may be cleared up by the following table. Identical terms are defined differently by authorities under the By column. As far as possible, tables and columns in the document are enclosed in square brackets, such as [t_object] or [Name] to make it clear when the database is used.

0	Term	By	Type	Definition
1	Abstract	OOP	Definition	An abstract type is an object type that cannot be instantiated directly. Abstract types are also known as existential types. This is used in [t_object]. Unused.
2	Attribute	11179	Standard	This is a characteristic of an object or set of objects.
3	Attribute	DAMA	Definition	Any detail that serves to qualify, identify, classify, or express state of an entity.
4	Attribute	Sparx	Definition	A row on the [t_attribute] table.
5	Blob	ML	BlobType	An object that does not have attributes. That is, an object without structure. It still has properties.
6	Class	OOP	Definition	A class is an extensible program-code-template for creating objects. A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behaviour (member functions or methods).
7	Class	Sparx	Object Type	A class is the most common object type. It is defined in the [t_objecttypes] table and is used on the [t_object] table. To add to the confusion, class reappears within Sparx as a UML class, where it is redefined as an OOP Class.
8	Concrete	OOP	Definition	A concrete type is an object type that can be instantiated directly. Unused.
9	Connector	Sparx	Object 2 Object	A row on the [t_connector] table. This is the primary means for defining object 2 object relationships. This was not renamed in the GUI layer.
10	Element	11179	Standard	An element or data element is a basic container for data.
11	Element	Sparx	Object	This is a row on [t_object]. This is renamed in the Sparx OO GUI. In the documentation, wherever Element is mentioned this refers to the [t_object] table.

0	Term	By	Type	Definition
12	Entity	DAMA	Definition	An entity may be defined as a thing capable of an independent existence of interest to the business that can be uniquely identified.
13	Enum	Computer	Definition	An enumeration of a sum type. This can also be called a code, such as Public or Private in Scope. Not to be confused with a program language like C#.
14	Feature	Sparx	Attribute	This is a row on [t_attribute]. This is renamed in the Sparx OO GUI. In the documentation, wherever Feature is mentioned this refers to the [t_attribute] table.
15	Glass	ML	BlobType	An object that does have attributes. That is, an object with structure. It also has properties.
16	Graph DB	Computer	Definition	A graph database is a database that uses graph theory node and edge data type tables to represent data. Graph data type tables are sufficiently abstract to represent any architectural diagramming method, such as data models, Zachman diagrams, etc.
17	Invariant	Computer	Definition	An expression whose value doesn't change during program execution. These are often used as properties for testing.
18	ISO 11179	ISO	Standard	A standard for representing metadata.
19	Key Value Tuple	Computer	Definition	A tuple with 2 elements. The pair consists of an identifier or key and a dependent variable or value.
20	Metadata	DAMA	Definition	Metadata is "data that provides information about other data". In other words, it is "data about data".
21	Model	Sparx	Object Type	This is a package which does not have a parent. That is, the [Parent_ID] = 0. There is only one called Welfare Data Model Repository. This is a related set of connectors, elements (objects) and features-(attributes). See Package, Root.
22	Object	OOP	Definition	An object is an instance of a class that contains properties and methods.
23	Object	Sparx	Definition	An object is a row in the [t_object] table. Sparx objects only have properties and do not have methods. They are used for defining a thing with an identity. There are 3 main object types: Class, Package and Text. To add to the confusion, object reappears within Sparx as a UML object, where it is redefined as an OOP Object. Also renamed as Element in the Sparx GUI.
24	One to Many	DAMA	Cardinality	Relation between 2 sets. Also written 1:M.
25	One to One	DAMA	Cardinality	Relation between 2 sets. Also written M:M.
26	Package	Sparx	Object Type	A package is both a row on the [t_package] and the [t_object] table. A package is used to group Sparx objects, diagrams, etc.
27	Package, root	Sparx	Object Type	This is a package which does not have a parent. That is, the [Parent_ID] = 0. There is only one called Welfare Data Model Repository. It contains a model. See Model.
28	Package,	Sparx	Object	This is a package which does have a parent. That is,

0	Term	By	Type	Definition
	view		Type	the [Parent_ID] <> 0. See the PROP pattern.
29	Project	Sparx	Group	This is a group of one or many Sparx models.
30	Property	11179	Standard	A characteristic common to all members in an object class.
31	Property	General	Definition	Something that belongs to something.
32	Property Value Tuple	Computer	Definition	Equivalent to Key Value tuple.
33	Property Value Tuple	Sparx	Pattern	These exist in the [t_attributetag] and [t_objectproperties] tables. Other tables can have this pattern, but it is not used. Critical for reification.
34	Reify	Computer	Definition	To take something that is abstract and make it material. For example, using data to create objects.
35	Repository	Sparx	Database	This is a database that contains all the Sparx data. It can cover multiple projects.
36	Sparx	Sparx	System	Sparx Systems Enterprise Architect is a visual modelling and design tool.
37	Stereotype	Sparx	Sparx	The most important typing of objects and attributes. It occurs as a column on the [t_object] and the [t_attribute] table. Not to be confused with the unused [t_stereotype] table.
38	Subject Area	DAMA	Definition	A grouping of data model entities or database tables.
39	Table Set	ML	Definition	A group of tables. This term is used in place of the normal term Subject Area.
40	Tuple	Computer	Definition	A finite ordered list of elements.

References

References – Books

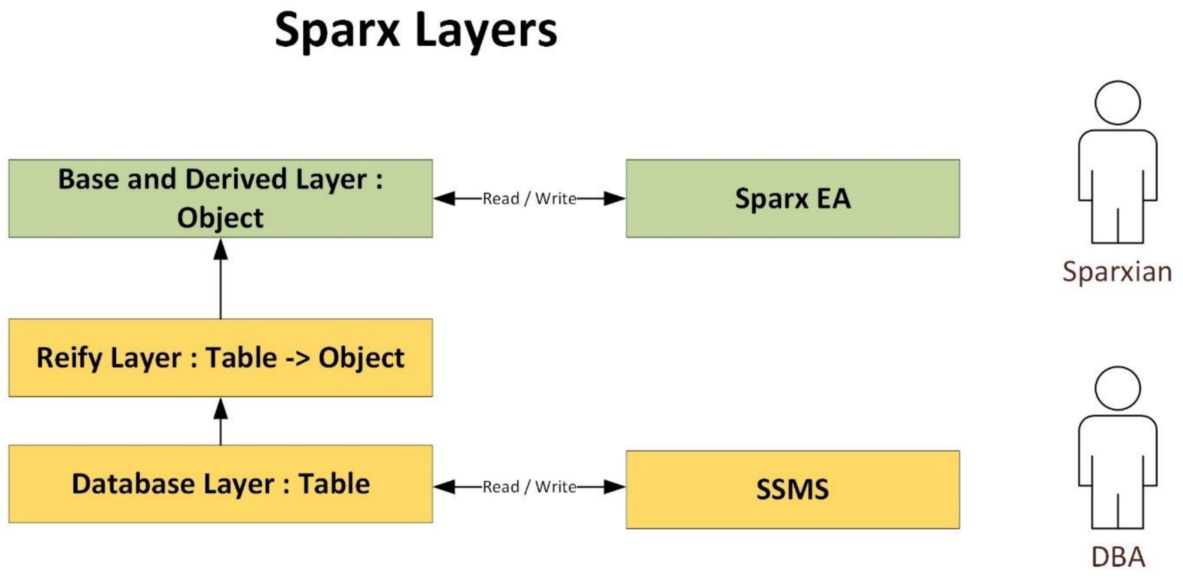
By	Name	For
Peter Doomen	Fifty Enterprise Architect Tricks	Usability guide
Sparx	Enterprise Architect Object Model	Guide to the repository objects for query or manipulation using an OO approach. It does not describe the underlying database.
Sparx	Model Navigation	Guide to navigate the EA Model.
Thomas Kilian	InsideEA	Sparx EA database.
Thomas Kilian	Scripting Enterprise Architect	Sparx EA scripting.

References – Web

By	For	Path
Geert Bellekens	Tips	https://bellekens.com/
Geert Bellekens	SQL	https://bellekens.com/#SQL
ISO	11179	https://www.iso.org/obp/ui/#iso:std:iso-iec:11179:-1:ed-3:v1:en
Sparx	Community	https://community.sparxsystems.com/
Sparx	Forum	https://www.sparxsystems.com/forums/smf/index.php

Application Layers

That is, how many Turtles are there? Under the hood, Sparx is no ordinary database.



User	Interface	Layer Name	Type	Description
Sparxian	Sparx EA	Derived Layer	Object	This shows all denormalised objects that depend on the layer below.
	Sparx EA	Base Layer	Object	This shows the base, normalised objects defined in the base packages.
	none	Reify Layer	Table -> Object	This shows the patterns used to create the objects in the layers above.
DBA	SSMS	Database Layer	Table	These are the raw Sparx DB Tables.

So, the Reify layer presents design objects to a Sparxian. This Reify layer uses design patterns that convert raw data into table like objects with relations and columns. These patterns need to be understood before integration data can be inserted into the database. Sparx now has a web layer sits above the client layer. This has not been examined.

DB Table Layer

From this point on, almost all tables are generated from the Sparx database itself. The SQL to create these tables is in the appendix. These tables are also saved into a separate excel spreadsheet.

DB Tables

Altogether there are 99 tables defined by Sparx. However, only 18 or about 20% are actively used much. The table below shows those names, short names, and row counts. The short names are introduced here, as they are used in the SQL examples in this document. These are all standard Sparx tables, and custom created tables are not used. All row Counts are from the sample and will vary with implementation.

Table Name	Short Name	Table Set	Row Count
t_attribute	TA	Reify	3,849
t_attributetag	TAT	Reify	27,765
t_cardinality	TCD	Reify	7
t_connector	TC	Reify	1,009
t_connectortypes	TCT	Reify	30
t_datatypes	TDA	Unused	645
t_diagram	TD	Diagram	74
t_diagramlinks	TDL	Diagram	1,108
t_diagramobjects	TDO	Diagram	1,854
t_diagramtypes	TDT	Diagram	15
t_document	TDC	Diagram	4
t_object	TOB	Reify	8,991
t_objectproperties	TOBP	Reify	43,587
t_objecttypes	TOT	Reify	80
t_package	TP	Reify	76
t_statustypes	TST	Diagram	5
t_stereotypes	TS	Unused	122
t_xref	TX	Reify	16,105

DB Relations

About half of the relations are conventional relations, which were discovered through profiling. That is, the parent column name and child column name are the same or similar. However, this naming convention is not consistent, and there are some false positive relations, which are called Fake in the table below. The relations are grouped into the table sets of Diagram and Reify. The term table set is used to describe what would normally be called a Subject Area, but this term has been redefined within Sparx. Note the high relation row count between the main Reify tables of [t_object], [t_objectproperties], [t_attribute] and [t_attributetag]. This is where the metadata lives (or hides).

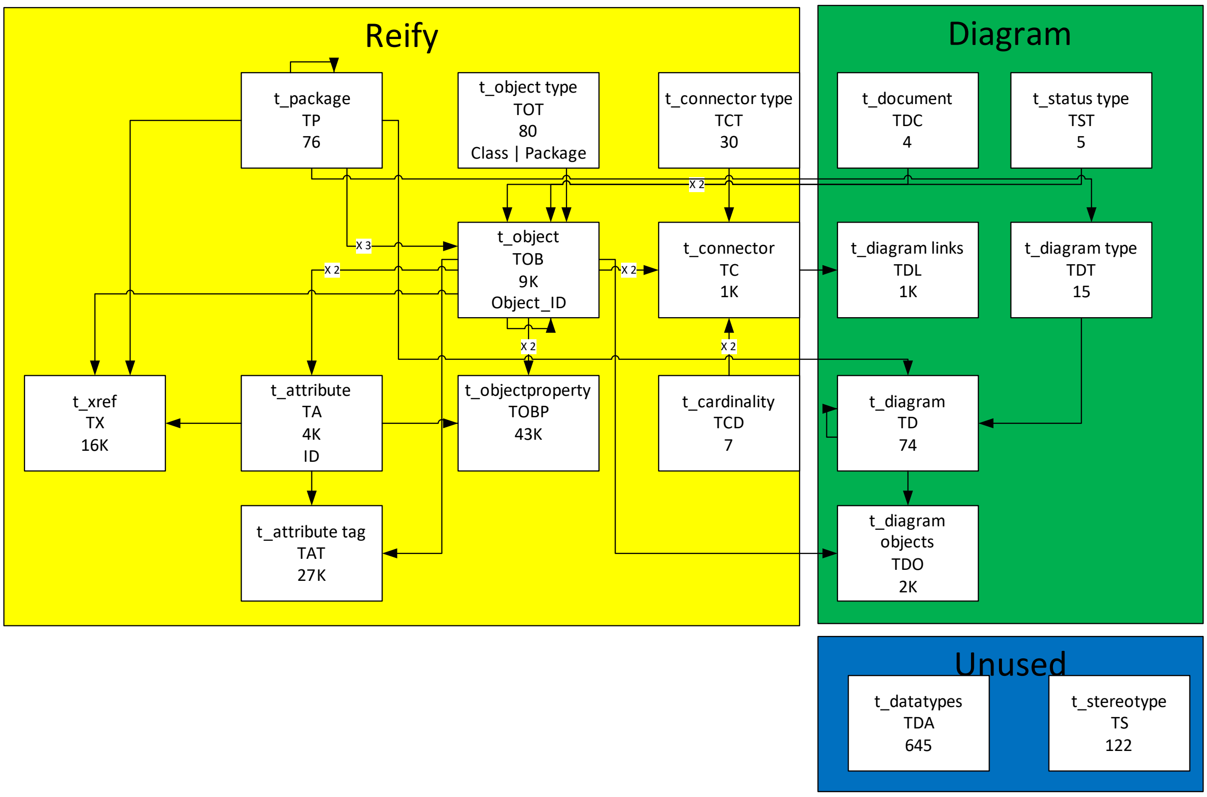
Parent Table	Parent Column	Child Table	Child Column	Row Count	Table Set
t_objecttypes	Object_Type	t_object	Object_Type	8991	Reify
t_object	Object_ID	t_objectproperties	Object_ID	43587	Reify
t_object	ea_guid	t_xref	Client	8946	Reify
t_object	Object_ID	t_attribute	Object_ID	3849	Reify
t_attribute	ID	t_attributetag	ElementID	27765	Reify
t_attribute	ea_guid	t_xref	Client	6157	Reify
t_object	ea_guid	t_objectproperties	VALUE	8543	Reify

Parent Table	Parent Column	Child Table	Child Column	Row Count	Table Set
t_attribute	ea_guid	t_objectproperties	VALUE	3069	Reify
t_attribute	ea_guid	t_attributetag	VALUE	1725	Reify
t_object	ea_guid	t_attributetag	VALUE	2114	Reify
t_cardinality	Cardinality	t_connector	DestCard	598	Reify
t_cardinality	Cardinality	t_connector	SourceCard	598	Reify
t_connectortypes	Connector_Type	t_connector	Connector_Type	1009	Reify
t_object	Object_ID	t_connector	End_Object_ID	1009	Reify
t_object	Object_ID	t_connector	Start_Object_ID	1009	Reify
t_connector	ea_guid	t_xref	Client	1002	Reify
t_object	Object_ID	t_attribute	Classifier	246	Reify
t_package	Package_ID	t_object	Package_ID	8991	Reify
t_package	Package_ID	t_object	PDATA1	78	Reify
t_package	Parent_ID	t_package	Package_ID	75	Reify
t_package	ea_guid	t_object	ea_guid	75	Reify
t_object	ea_guid	t_package	ea_guid	75	Reify
t_object	ParentID	t_object	Object_ID	7165	Reify
t_stereotypes	Stereotype	t_attribute	Stereotype	0	Fake
t_diagram	Diagram_ID	t_connector	DiagramID	0	Fake
t_stereotypes	Stereotype	t_connector	Stereotype	0	Fake
t_diagram	ParentID	t_diagram	Diagram_ID	0	Fake
t_stereotypes	Stereotype	t_diagram	Stereotype	0	Fake
t_diagram	Diagram_ID	t_object	Diagram_ID	0	Fake
t_stereotypes	Stereotype	t_object	Stereotype	2	Fake
t_image	ImageID	t_objecttypes	ImageID	0	Fake
t_object	ea_guid	t_attribute	ea_guid	0	Fake
t_attribute	ea_guid	t_attributetag	ea_guid	0	Fake
t_object	ea_guid	t_attributetag	ea_guid	0	Fake
t_object	ea_guid	t_diagram	ea_guid	0	Fake
t_object	ea_guid	t_xref	Link	0	Fake
t_object	ea_guid	t_xref	Supplier	0	Fake
t_diagramtypes	Diagram_Type	t_diagram	Diagram_Type	74	Diagram
t_package	Package_ID	t_diagram	Package_ID	74	Diagram
t_connector	Connector_ID	t_diagramlinks	ConnectorID	1108	Diagram
t_diagram	Diagram_ID	t_diagramlinks	DiagramID	1108	Diagram
t_diagram	Diagram_ID	t_diagramobjects	Diagram_ID	1854	Diagram
t_object	Object_ID	t_diagramobjects	Object_ID	1854	Diagram
t_package	Package_ID	t_diagramtypes	Package_ID	15	Diagram
t_statustypes	Status	t_object	Status	8989	Diagram
t_document	DocID	t_object	ea_guid	3	Diagram
t_document	ElementID	t_object	ea_guid	3	Diagram

Table Sets

The table sets are Reify, Diagram and Unused in the Sparx DB data model diagram below. The relations within the Diagram table set follows typical relational database patterns. Consequently, these are visible through multiple layers. However, the relations within the Reify table set do not. Explaining that is the core purpose of this document.

Sparx: Table Sets: Reify | Diagram | Unused



Diagram

In general, these tables are only used for the Sparx GUI presentation layer. They are not used to define metadata objects. They follow standard relational patterns, which are discussed below.

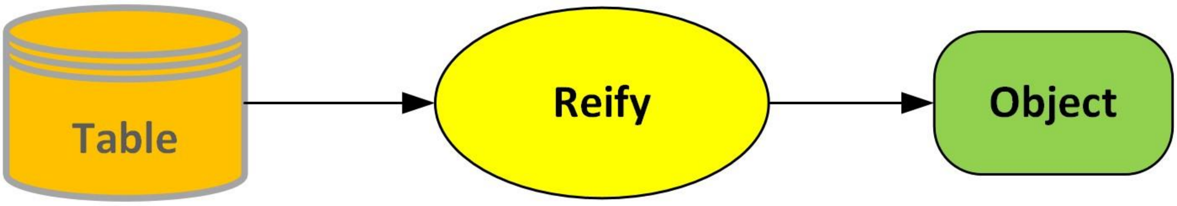
Unused

There are many empty Sparx tables in the example, so these are called unused. There are 2 potentially useful tables which have many rows but are not used. These are [t_datatypes] and [t_stereotypes]. [t_datatypes] contain physical DDL or physical table types. [t_stereotypes] contains many apparently useful types for metadata usage, that are not used.

Reify

To "reify" something is to take something that is abstract and regard it as material. Another way to define is that it brings something to life. In this case, the data is used to create objects. This is the flat pack view of the database. Reification can be thought of as the Allen key that turns the flat pack into a shelf or a desk. Reification is an important topic in Computer Science.

Table to Object



There are 4 main tables that create objects in the object layers. These are [t_object], [t_objectproperties], [t_attribute] and [t_attributetag]. All 4 tables are central to definition of relationships between reified objects.

Also, within Reify, there are some other tables. These are [t_objecttypes], [t_package] and [t_connector].

The [t_objecttypes] table has the highest categorisation of objects. Out of 80 choices, only 3 are used. These are Class, Package and Text. All primary objects have an object Type of Class.

The [t_connector] table is used for one of the Object 2 Object relations patterns, discussed below.

[t_package]

The [t_package] table is the major means of determining object containers. There are 5 main types of package. Packages are in a package hierarchy. They are determined by the parent_id. See table below. Most Packages are also defined in the [t_object] table. The Packages use all 3 object types: Class, Package and Text.
Objects are linked to packages using the package_ID column. Some packages allow objects with different stereotypes, so there is no stereotype checking as there would be with collections. The packages are often organised into a hierarchy using the Parent_ID column.

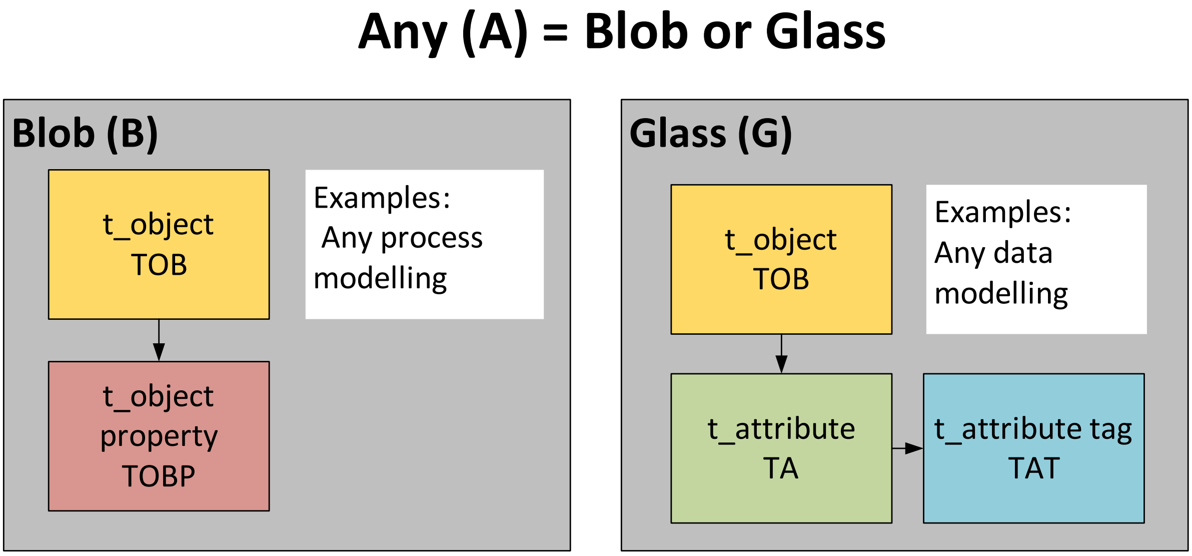
Almost all stereotypes belong to a package. However, there can be orphan Stereotypes. The packages are typically organised into a simple 3 level tree, with grandparents, parents, and children.

Blob and Glass objects

Whether an object has attributes or not drives important reification pattern choices within Sparx EA. To separate this characteristic, I introduce a new object type which can be either Blob or Glass. A Blob is an object without attributes, and a Glass is an object with attributes. The reasoning will become clear as the reification patterns are explained below.

Another way to think of a Blob is as a single, indivisible object without parts or attributes. A Glass is a transparent object with parts which are defined using attributes. This means that a Glass is also a collection or list object. Note that an object can be defined as a collection object, but if it does not have attributes, then the collection is undefined. Each Stereotype is either a Blob or a Glass. Note that a Blob object still needs to be identified with an [Object_ID] and a [Name], but each Blob only has one of these properties, not many. Naturally, a Glass object also has an [Object_ID] and a [Name], but it also has many attributes, which expose the internal structure.

A key example or use case is when defining database metadata. A Glass object is needed to define a DB schema table. That is, the object represents the schema table. Each object property are properties of the table, and not of the table columns. The child attribute is a row. Each row defines a single table column. The attribute tags represent the properties of a single column like name, position, nullability, physical data type, etc. A Blob object is not sufficient to define a DB schema table, as table columns would be undefined. Another example is when defining a process. In this case, a Blob object would be just fine. In the simple case, all that is needed is an id, a name, and a description.



Why are Blobs used? In practice, they are the primary concept in Sparx EA, so most objects are Blobs. Glass objects are more refined than Blob objects, so these are only defined when needed. Blob objects are used when only coarse analysis is needed, but when more refined analysis is needed, then Glass objects are used.

In the example, there are only three Stereotypes that are Glass objects. An example might be LookupData. If the Object Stereotype is LookupData, then the tuple (table Name, Attribute Name) is a unique Attribute PK. If the Object Stereotype is something else, then the tuple (table Name, Attribute Name) is non-unique Attribute PK, and (Package Name, table Name, Attribute Name) is a unique Attribute PK. Knowing these key invariants helps to understand the relations described in DROP below.

Reify Table Set

Reification Object Patterns

There are a variety of patterns which reify tables into the many objects of the object layer. There are 5 kinds of reification object patterns: Type, Column, Parent, DAG, and Graph.

TROP – Type Reification Object Pattern

These are implied Enum or code values. As they are not captured as separate tables, they are identified as distinct value set from specific columns such as [Stereotype] and [Property].

There are 4 TROP Patterns: Object Stereotype, Object Property, Attribute Stereotype and Type and Attribute Tag Property. There are other types within Sparx, but these are not used in the example.

TROPB1: Object Stereotype

All objects have a Stereotype, which defines the type of object. The [t_object] [Stereotype] column is the most important, as this is the primary type for all Sparx objects. The [Stereotype] column has a nvarchar type. So, any string can be added to these columns, without DB type checking.

The most important Object Types are Class and Package. The example has 4 Package blobs and 17 Class blobs. The other Object Types are not used. Again, it is noted that these Stereotypes are not defined in any other table, including the [t_stereotype] table. TROPB2: Object Property

The [t_objectproperties] Property column is the next most important type, as these determine common or boilerplate properties on almost all Objects. Each object has only one instance of a property, such as Status, Security Classification, Governance Entity, etc. Obviously, objects already have a fixed number of properties, that are physical columns on the [t_object] table, such as [Object_ID], [Name], [Author], [Note] and [Stereotype]. Each [t_objectproperties] [Property] is an additional, customisable property for the object specific to each [t_object] [Stereotype].

These rules only apply to [t_objectproperties] when the [Value] string that is not a GUID. That is, a GUID is a string that starts with "{". [Value] which contain a GUID are used to reify relations, which are explained in the DROP patterns below. See a GUID regex in the appendix.

From the list of names, many standard properties are not available OOB. But this approach does support any additional needed property is obvious.

Clearly, one the dangers of this method of defining additional properties on objects is that there is either minimal or no type safety. The default type for these properties is string, and decimal, date or simple enum data types can be used. This kind of type checking would need to be introduced later in using a trigger or post load SQL script.

Object_Type	Property	CT
Class	Colour	25
Class	Data Type	7482
Class	derivation	2811
Class	Dissemination Classification	291
Class	Form Identifier Code	1
Class	fraction Digits	42
Class	Implements	8193
Class	Industry Standard	217

Object_Type	Property	CT
Class	Item Type	7482
Class	Level 1	8
Class	Level 2	8
Class	Level 3	8
Class	Level 4	8
Class	Max Length	2737
Class	Max Occurs	3840
Class	minOccurs	3840
Class	model Group	596
Class	pattern	41
Class	Published Date	1
Class	Security Classification	291
Class	Status	291
Class	Subject Area	291
Class	total Digits	72
Class	Version	1
Package	Data Stewardship Contact Email	11
Package	Source Location	19
Package	Source Type	19
Package	Source Version	19
Package	Tile image	11

This table shows the counts for some important properties and their values. This should make it clear their usage. Again, the enum types are not checked.

Property	Value	CT
Data Type	date	335
Data Type	decimal	72
Data Type	string	2834
Data Type	unsignedByte	4
Dissemination Classification	Official use only	508
Governance Entity	Australian Securities and Investments Commission	1
Governance Entity	Australian Taxation Office	7
Industry Standard	ABS SEIFA	1
Industry Standard	AS4590	10
Security Classification	Protected	508
Status	Deprecated	1
Status	Development	2
Status	Draft	25
Status	Proposed	538

TROPG1: Attribute Stereotype and Type

What is the difference between [t_attribute] and [t_objectproperties]? Each Object can have many values of the same [t_attribute] [Stereotype]. That is, the [t_attribute] is a child table, and that it represents a one-to-many relationship. Each [t_object] can only have one [t_objectproperties]. That is, only one instance of Status, Security Classification or Governance Entity.

These [t_attribute] types are the next most important, as these determine attribute types for some Objects. [t_attribute] are only populated for a few object Stereotypes, as these are only needed for special cases. Examples of uses cases are LookupData or Entity type objects. A [t_attribute] can be thought as the Key of a KV tuple. A [t_attributetag] can be thought as the Value of a KV tuple.

The table shows the cross product of [Stereotype] and [Type] columns from the [t_attribute] table. It can be seen that [Stereotype] on [t_attribute] is reasonably consistent, with only a few errors. [Type] on [t_attribute] is a mix up of code and primitive data types. Again, there is a lack of type safety and [Stereotype] and [Type] are only loosely related. [Stereotype] = Attr_Text sometimes links to a Code or type. Finally, when [Stereotype] = Attr_Code, then [Type] seems to have correct values, except when [Type] = string.

Stereotype	Type	CT
Attr_Amount	Amount	53
Attr_Amount	decimal	199
Attr_Count	Count	26
Attr_Count	integer	77
Attr_Date	date	198
Attr_Date	dateTime	25
Attr_Date	string	1
Attr_Datetime	Datetime	27
Attr_Identifier	bigint	1
Attr_Identifier	string	72
Attr_Indicator	boolean	399
Attr_Number	integer	89
Attr_Number	Postcode Type	1
Attr_Number	string	2
Attr_Percent	decimal	21
Attr_Percent	integer	1
Attr_Text	NULL	5
Attr_Text	boolean	1
Attr_Text	Country Type	2
Attr_Text	Employment Paid Leave Type	1
Attr_Text	Party Identifier Type	1
Attr_Text	string	374
Attr_Text	Suburb Type	1
Attr_Time	time	1
Attr_URL	string	6

TROPG2: Attribute Tag Property

These [t_attributetag] values are the least important, as these populate only some attribute properties. As the Value of KV tuple, each [t_attributetag] is only related to one [t_attribute].

Again like [t_objectproperties], the one-to-one rule only applies to [t_attributetag] with a [Value] string that is not a GUID. That is, a string that starts with "{". [Value] which contain a GUID are used to reify relations, which are explained in the DROP patterns below. These value driven rules can make the design inexplicable.

There are duplicate properties for both object and attribute. These are Security Classification, Dissemination Classification, Guidance, Status, etc. In the case of Security Classification, Dissemination Classification, Guidance and Status this duplication is fine, because these properties can be different on the Object and the Attribute.

This table shows the counts for some important properties and their values. This should make it clear their usage.

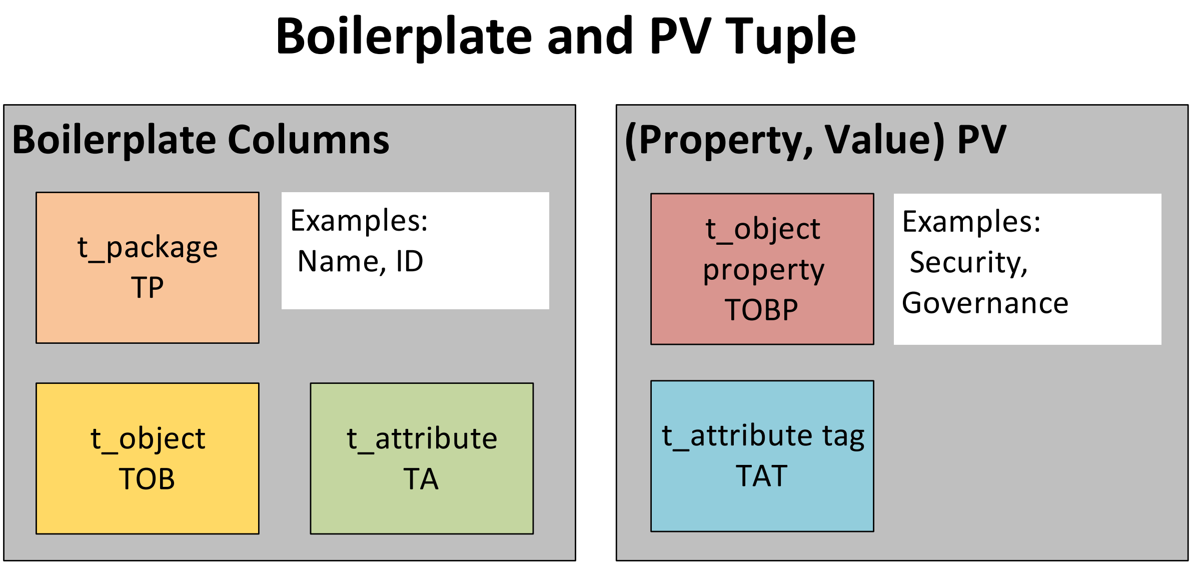
Property	VALUE	CT
Dissemination Classification	Official use only	2114
Dissemination Classification	Sensitive personal	1
Security Classification	Protected	2115
Status	Approved	1
Status	Development	23
Status	Draft	250
Status	Proposed	1841

CROP – Column Reification Object Pattern

This is how columns of an object are defined.

There are two pattern types. These are Boilerplate and Property Value. The patterns are applied twice. The first is the object or table level, and second is the attribute or column level.

From this, there are two Column Reification Object Patterns called CROP, which apply to Blob and Glass objects, which are called CROPA and CROPG. In each pattern, columns are reified as Boilerplate columns, and others are reified from a child Property Value table.



Boilerplate Pattern Type

Each core table has 2 kinds of boilerplate columns; columns usable for metadata, and other columns that are unused. The unused columns support Sparx features which may be compromised if reused for other purposes. The conservative approach is to avoid these. These unused columns will not be documented below, but they can be inferred by checking the difference between the available columns and the RDBMS schema. See the appendix for a table covering the usage for all Boilerplate columns.

The boilerplate tables are [t_package], [t_object] and [t_attribute]. These all contain reusable boilerplate columns. Note that [t_objectproperties] and [t_attributetag] do not contain boilerplate columns, so these are not used in this pattern. Instead, they are used in the CROPA/CROPG pattern below.

A package can be any kind of object collection. As a package is also an object, the object boilerplate columns are also available to define a package.

Property and Value (PV) Pattern Type

Both [t_objectproperties] and [t_attributetag] are PV tables, where the [Property] and [Value] are both just strings. This means that any Property name, and any Value can be created. The Property is useful, common information about the objects and attributes in general, such as Security, Status, Contact emails, etc.

This pattern uses the same PV tables as in the TROP3 and TROP4 patterns above. But this pattern is about reifying to columns rather than types. Again, these rules only apply to [t_objectproperties] and [t_attributetag] with a [Value] string that is not a GUID. That is, a GUID is a string that starts with "{". Properties that reify into Foreign Keys FK are defined below in the DROP patterns.

These type values are not database type checked of course. As the Value column has a length of 255, then all strings, and other values, have this characteristic. That is, maximum flexibility and zero type safety.

This is about reusing current properties as much as possible. Naturally, it is trivial to add new Properties to [t_objectproperties] or `[t_attributetag]`.

The following table shows the property names that have already been used, and the sample values. The table shows current defined properties, their inferred type, length, sample or type values, and table used. Whether the table is [t_objectproperties] or `[t_attributetag]` is not important, as these properties can be used on either table with any Stereotype. Again, there is maximum flexibility and minimum type safety.

Property	Sample Value	PV Table
Author	Joseph Gillespie	t_objectproperties
Colour	10216385	t_objectproperties
Contact Email	x @y.gov.au	t_objectproperties
Data Type	string	t_objectproperties
Dissemination Classification	Official use only	t_objectproperties
Document Date	6/09/2019	t_objectproperties
Exchange Reference	http://xyz	t_objectproperties
fraction Digits	2	t_objectproperties
Governance Entity	Australian Taxation Office	t_objectproperties
Industry Standard	AS4590	t_objectproperties
Internal or External	External	t_objectproperties
Max Length	1	t_objectproperties
Max Occurs	1	t_objectproperties
minOccurs	1	t_objectproperties
Order	1	t_objectproperties
pattern	\d+	t_objectproperties
Security Classification	Protected	t_objectproperties
Source Code File	report.xls	t_objectproperties
Source Code Location	path.xyzda.gov.au/ folder/	t_objectproperties
Source Version	Not applicable	t_objectproperties
Status	Proposed	t_objectproperties
Tile image	id=487436947;mdg=Global;	t_objectproperties
Total Digits	8	t_objectproperties

Property	Sample Value	PV Table
Dissemination Classification	Official use only	t_attributetag
External Source of Truth	ATO ITR Exchange	t_attributetag
Internal Source of Truth	DHS Data Lake	t_attributetag
Is Business Key	FALSE	t_attributetag
Max Length	3	t_attributetag
Optionality	Optional	t_attributetag
Precision	2	t_attributetag
Security Classification	Protected	t_attributetag
Status	Proposed	t_attributetag
Synonym	DoE	t_attributetag

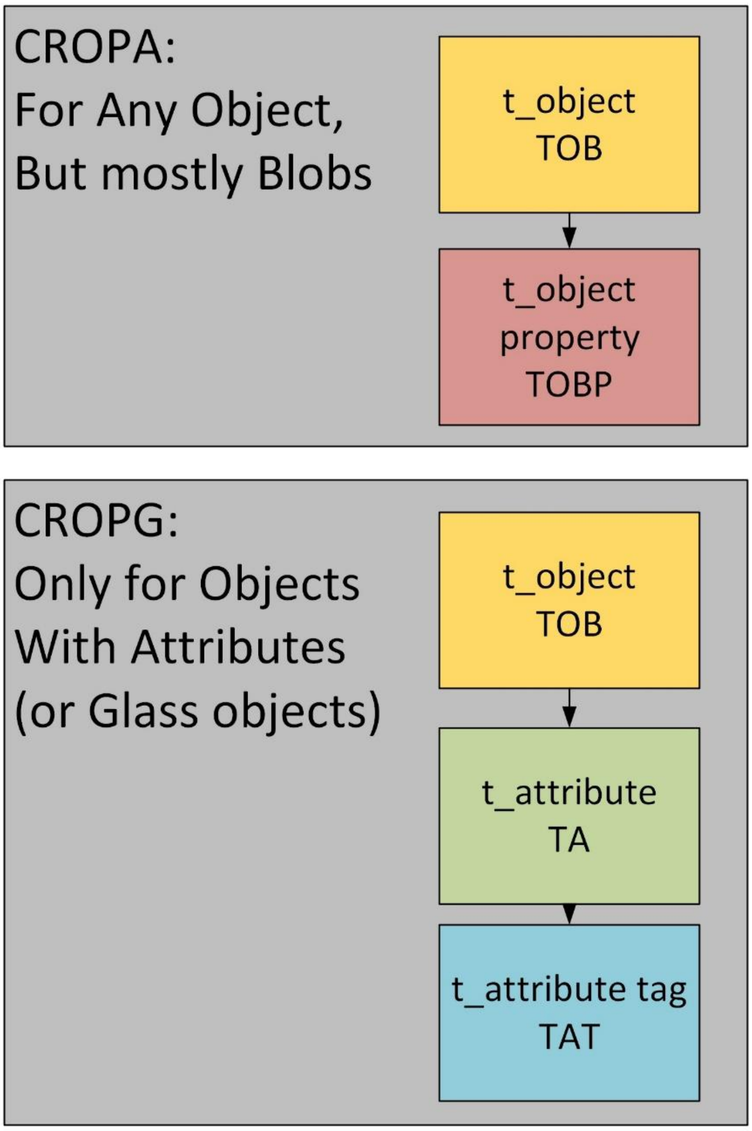
CROPA: Object and Object Property

This pattern applies to the [t_object] and [t_objectproperties] tables. In this pattern, there is no need for the [t_attribute] table. Therefore, it is called an Any pattern, as it mainly applies to Blob objects, but it also can apply to Glass objects as well. This applies the PV pattern above to [t_objectproperties] and the [t_object] table to give more detailed information about the object. There is a one-to-one relationship rule between [t_object] table and the [t_objectproperties] table. There can only be one Property type for each object. For example, an object can only have 1 colour.

Nullability is defined by the absence of a value. This can be done by allowing [Value] column to be NULL, or by not having a [t_objectproperties] row for the object. Conversely, if all [t_object] rows have a [t_objectproperties] row, where the [Value] column is not NULL, then this would indicate a mandatory or not nullable column.

Again, these rules only apply to [t_objectproperties] with a [Value] string that is not a GUID. That is, a GUID is a string that starts with "{". Properties that reify into Foreign Keys FK are defined below in the DROP patterns.

Column Reify Object Pattern



CROPG: Attribute and Attribute Tag

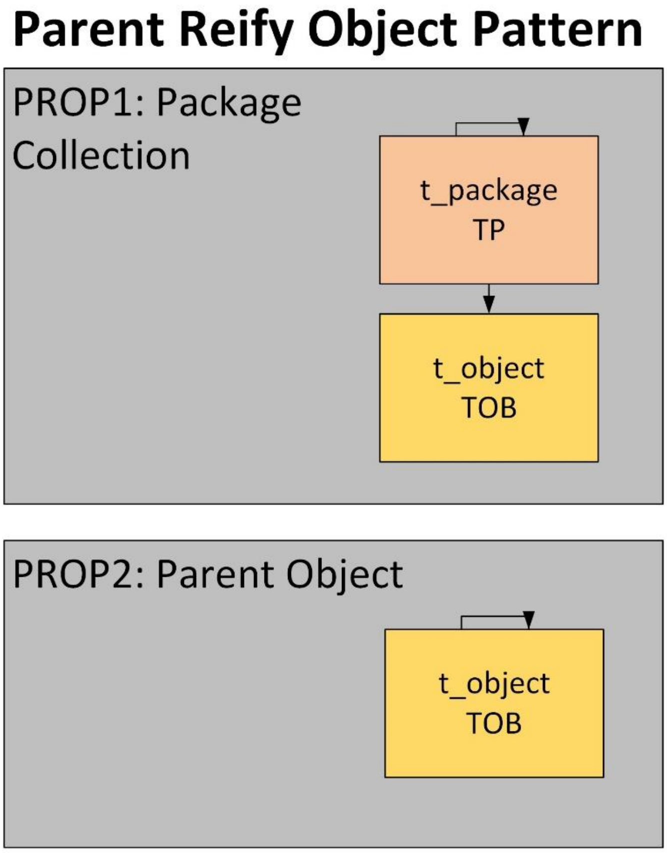
This applies the PV pattern type above to [t_object], [t_attribute] and [t_attributetag] to create a pseudo–DB Schema Column table. This will be a child of the DB Schema Table above. The Foreign Key FK is [Object_ID] and ID that points back to the DB Schema Table.

This DB Schema Column table is the table that is closest to the SQL Server [COLUMNS] table in the [INFORMATION_SCHEMA] which defines metadata for DB columns. As this is a common pattern, it is easy to reason about this metadata.

If the [t_attribute] does not have an [t_attributetag] child, then this is reified as a nullable column.

If the [t_attribute] does have an `[t_attributetag]` child, then the content of the [Value] column determines the columns behaviour or type. If the does not contain a GUID string, then it is a simple type. If it has a GUID string, then it will be a relation. There are two relation types which are discussed below in the DROP below.

PROP – Parent Reification Object Pattern



A PROP or Parent Relationship is a pattern that follows the standard relational single parent hierarchy relationship, which is implemented as a self-join on the same table. The key thing is that this pattern applies to the same table.

These relationships rely on the primary key [Object_ID]. This contrasts with the DROP pattern below, where all the relationships rely on the alternate primary key [ea_guid]. There a few important examples used in the metadata modelling.

PROPA: Package

The [t_package] table has a single parent column: `[Parent_ID].` The [t_object] table has a join to package using [Package_ID]. This represents the object for the package. The [t_object] table also has a join to the parent package using [PDATA1]. All columns are critical in creating a self-join hierarchy or object tree. The tree can consist of any number of packages and objects. In practice, it is more limited. Using these columns, a hierarchy of packages can be defined, from grandparent to parent to child, which link to single leaf objects. There are also orphan objects that do not belong to a package.

PROPA: Object

The [t_object] table also has a single parent column: `[ParentID].` This object self-join is used only by one example which models physical DB columns.

ROP Type	Child_OT	Child_ST	Child2Parent	Parent_OT	Parent_ST	CT
PROPA2	Class	Column Type	isChildOf	Class	Column Type	7140

PROPA: Object Classifier

The [t_object] table also contains 2 other potential single parent columns: [Classifier] and [Classifier_guid]. These columns are not used at all, as they always NULL. It is assumed [Classifier] would self-join to [Object_ID] and [Classifier_guid] would self-join to [ea_guid]. Given the availability of [ParentID], use of these columns should be avoided.

DROP – DAG Reification Object Pattern

This is called the DAG or directed acyclic graph pattern. A DAG is often used to describe one to many relations between different objects. These relations all additional tables beyond [t_object]. These are the most complicated to define, as they do not follow regular or easily understood relational design patterns.

All these relationships rely on the alternate primary key [ea_guid]. This contrasts with the PROP pattern above, where all the relationships rely on the primary key [Object_ID]. Note that [ea_guid] is an alternate Primary Key on almost all tables. This provides a great deal of flexibility in joining tables, in that any table with a [ea_guid] column can be joined to any other table with an identical column. However, this flexibility has the downside of creating confusion or ambiguity about which table joins to which. This can only be discovered through profiling.

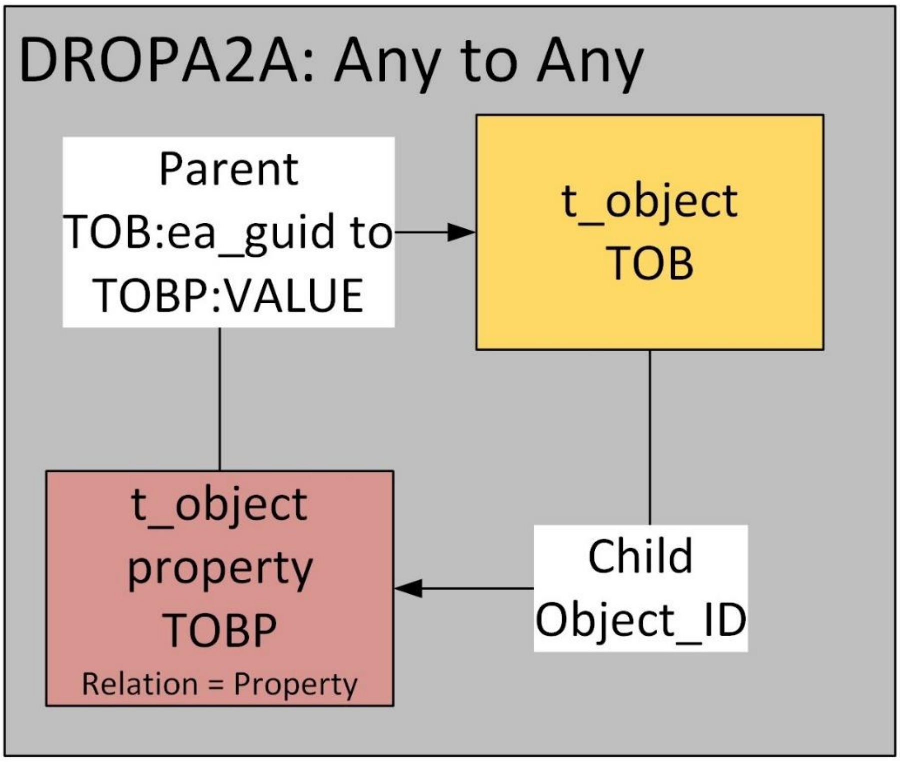
These rules only apply to [t_objectproperties] and the [t_attributetag] tables with a [Value] string that IS a GUID. That is, a GUID is a string that starts with "{". If so, then this row defines a relationship back to the [ea_guid] column on either a [t_object] or [t_attribute] table. Note that these relationships are only ever one to many (1:M). It is not possible in these patterns to define other cardinalities. If [Value] does not contain a GUID then they are used to reify types and columns, as covered in the TROP and CROP patterns above.

There are 3 kinds of DROP between objects: Any to Any, Blob to Glass, Glass to Any. There is another kind of DROP, but this links attributes for Column Data Types. The following table shows currently defined relations using the patterns and Stereotypes.

Most of the examples (12) uses DROPA2A pattern. 3 examples use the DROPA2G pattern, only 1 uses the DROPCDT pattern.

DROPA2A: Any to Any

DAG (Directed Acyclic Graph)
Reify Object Pattern 1

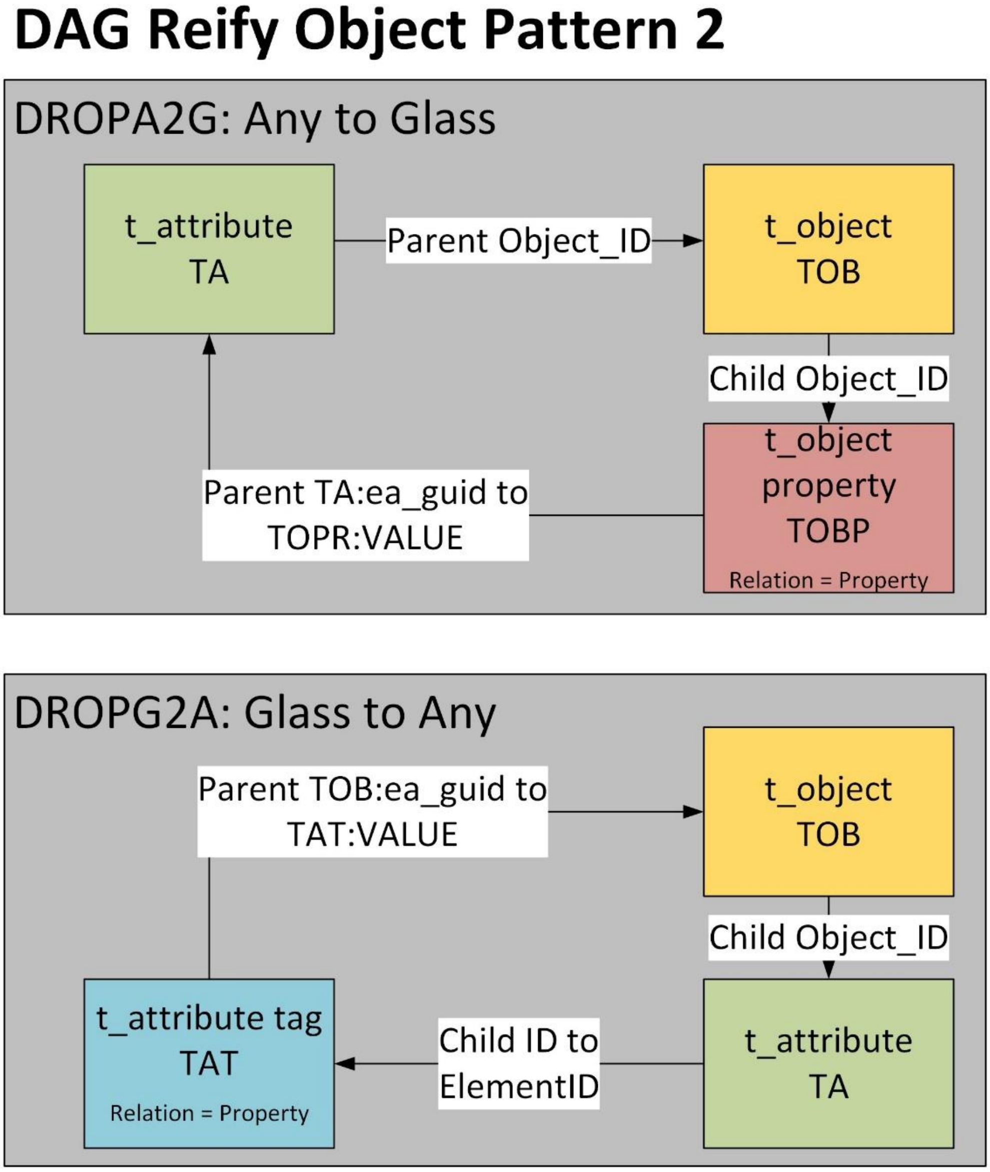


Even though it is called Any to Any, this is essentially used for Blob-to-Blob relations because only object information is needed. That is, both the parent and the child can be regarded as a Blob, and any attribute information is ignored. So, there is nothing like an FK column, and the PK is always the [ea_guid]. Obviously, if both objects are Blobs, then this is the only choice. Naturally this pattern is usable for Glass objects as well, but the other patterns are available to provide richer detail about the relationship such as FK name, etc.

This Object 2 Object pattern is a join between Object, object properties and back to object. The relation is defined as a link between objects, but it is not further defined. Because it is so simple, and can be used between any objects, including both Blob and Glass objects, this is the most common relation. Altogether this pattern is used 12 times.

For example, this pattern can be used for showing a DB hierarchy such as DB to Db Schema to DB Table to DB Column, etc.

DROPA2G: Any to Glass



This is mainly for relationships where a Glass object is the parent, and a Blob object is the child. Both examples are best avoided.

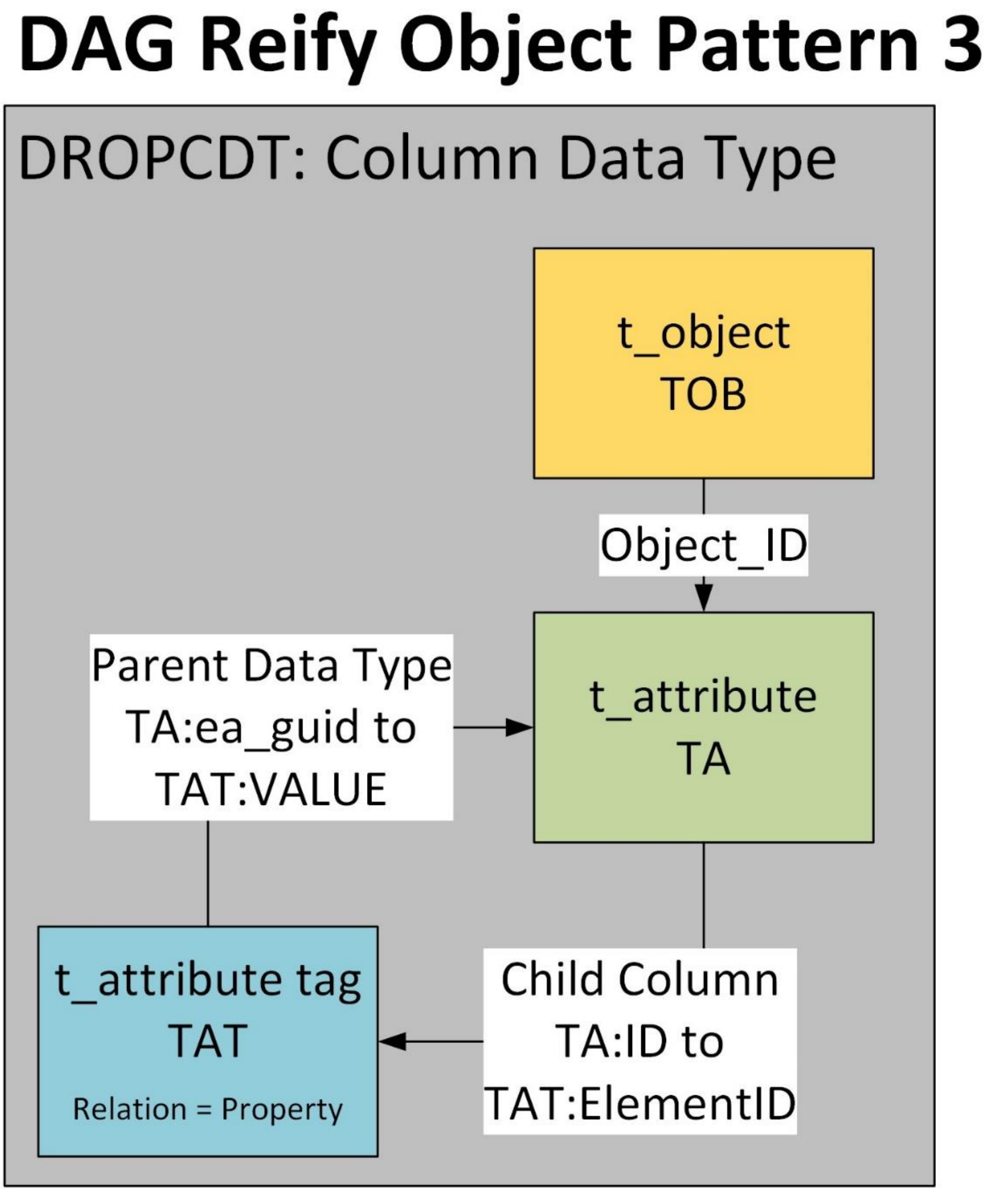
This Object 2 Object pattern is a join between Object, object properties, attribute and back to object. This could be called a Pseudo Foreign Key because it resembles a relational Foreign Key link. In this case, the object property of the child object is the relation’s name. The ea_guid Value of the object property then links to an attribute, and then to the parent object. An example of a relationship types is to link DB columns to a metadata attribute.

DROPG2A: Glass to Any

This is mainly for relationships where a Glass object is the child, and a Glass object is the parent. It is the best choice for relationships where both objects are Glass objects as it captures the relationship in the fullest detail. It could be called Standard Foreign Key because it reifies a relational Foreign Key link completely. From a relational DB perspective, this pattern defines best the foreign keys (FK) relation between tables. Clearly, the parent object could be a Blob, but in this case, it would be simpler to use the DROPA2A pattern.

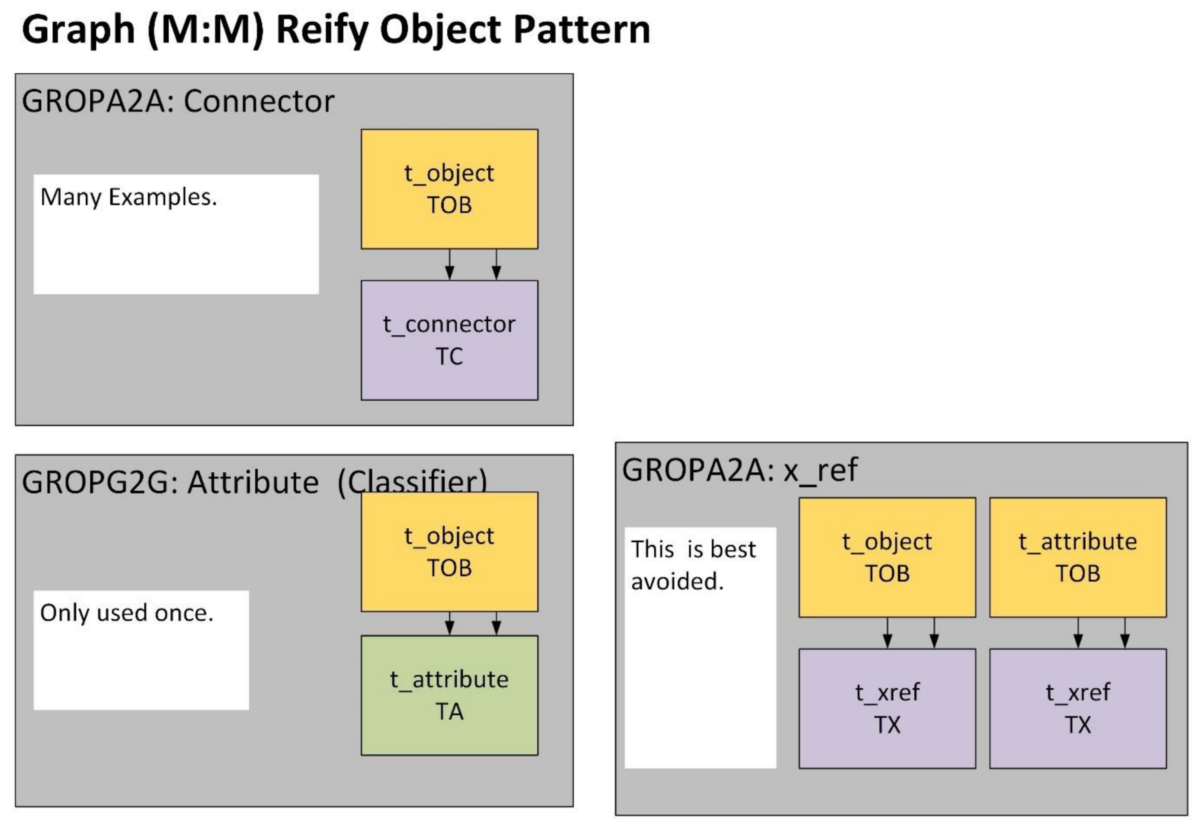
This Object 2 Object pattern is a join between Object, attribute, attribute tag and back to object. The child Glass object must have an attribute that behaves as Foreign Key column. The attribute tag then points to the ea_guid of the parent object. If the parent was a Glass object, then it would have a primary key column defined and this could be matched with the child attribute name.

DROPCDT: Column Data Type



This is a special DROP that links attributes together, rather than objects. The only use case is to reify the Column data type definition, hence the name. This pattern can only be used by Glass objects. This Attribute 2 Attribute pattern is a join from attribute-to-attribute tag and back to attribute. It is best avoided.

GROP – Graph Reification Object Pattern



These are generalised graph patterns, or many to many relations (M:M). The PROP and DROP patterns apply to one-to-many relations. These many to many relations can be decomposed into two one to many relations, as well.

GROPA2A: Connector

This can be used for any object-to-object relationship. It is the only many to many relationship available to Blob objects, as it does not need attribute level information.

This Object 2 Object pattern is a join between Object, connector and back to object. The `[t_connector]` table has 2 Object_Id columns: [Start_Object_ID] and [End_Object_ID]. This supports the standard relationship multi-parent hierarchy design pattern. It can define many to many relations between any 2 objects. Many of the relations have names such as ‘is a’, ‘may have’ or ‘must have’. [t_connector] is used to link objects on diagrams.

The connector types are restricted to Association, Generalization and ObjectFlow. These are defined on the [t_connectortypes] table. It may be possible to add additional connector types for custom created relationships. This table could also be used to create invariants for comparison to any generated or derived relations.

GROPG2G: Attribute Classifier

This can be used for mainly for Glass-to-Glass relationships. The only use case is a Classifier relation, given the column name. It may not be a Sparx design intention to expand the use of this to other more general relationships.

This Object 2 Object pattern is a join between Object, attribute and back to object. The [t_attribute] table has 2 Object_Id columns: `[Object_ID]` and `[Classifier].` Currently, this only captures one relationship.

Diagram Table Set

Even though the diagram tables do not define objects, they do control how visible these objects are in the Sparx presentation layer. So, the plan is to define how those diagrams can become visible. The goal is to safely integrate external metadata into Sparx, and to make it visible to a Sparx user. This is incomplete.

Other object relations could be created indirectly using the [t_diagramobjects] and `[t_diagramlinks]` tables. They could also be used for invariants. There has been little analysis on these currently.

Some tables need further comment. [t_statustypes] is used only to filter objects on diagrams. Other usages of status types will be covered in the TROP section. [t_connector] is discussed in the DROP section. [t_xref] is discussed in the appendix.

[t_xref] is an interesting table, but it is not used in the reification patterns above. [t_xref] uses ea_guids in the [Client] column, and these are used to join to parent tables such as [t_object], [t_attribute] and [t_connector]. It adds properties or attributes to these tables.

DRAFT Rule Book

This is preliminary rough first draft of the rule book. The table below shows the current implied rules that turn a relational concepts into an ROP or Reified Object Pattern. The rules will make sense as the document progresses.

O	For	ROP	Boolean	Ans	Choice
1	Type	TROP	Does thing types list already defined in TROPs	No	Define new TROPS
1	Type	TROP	Does thing types list already defined in TROPs	Yes	Reuse current TROPS
2	Column	CROPA	Are thing's properties already defined in object properties list?	No	Reuse object properties
2	Column	CROPA	Are thing's properties already defined in object properties list?	Yes	Create new object properties
3	Column	CROPG	Does thing have any rows	No	Choose Blob, and do not define Attributes
3	Column	CROPG	Does thing have any rows	Yes	Choose Glass, and define Attributes
4	Relation	PROP1	Does thing belong in a collection?	No	Ignore Package_ID
4	Relation	PROP1	Does thing belong in a collection?	Yes	Create or Reuse a Package_ID
5	Relation	PROP2	Does thing have a self-join hierarchy?	No	Ignore ParentID
5	Relation	PROP2	Does thing have a self-join hierarchy?	Yes	Use ParentID
6	Relation	DROPA2A	Is Child object a Blob and Is Parent object a Blob?	No	Go to 7
6	Relation	DROPA2A	Is Child object a Blob and Is Parent object a Blob?	Yes	Define DROPA2A relation
7	Relation	DROPA2G	Is Child object a Blob and Is Parent object a Glass?	No	Go to 8
7	Relation	DROPA2G	Is Child object a Blob and Is Parent object a Glass?	Yes	Define DROPA2G relation
8	Relation	DROPG2A	Is Child object a Glass?	No	Undefined
8	Relation	DROPG2A	Is Child object a Glass?	Yes	Define DROPG2A relation
9	Relation	DROPCDT	Is a column Data Type needed for a Column Attribute?	No	Undefined
9	Relation	DROPCDT	Is a column Data Type needed for a Column Attribute?	Yes	Define DROPCDT relation
10	Relation	GROPA2A	Does Child object need a relation that is not a DROP type?	No	Undefined
10	Relation	GROPA2A	Does Child object need a relation that is not a DROP type?	Yes	Define DROPA2A relation
11	Relation	GROPG2G	Does Child Glass object need a relation that is not a DROP type?	No	Undefined
11	Relation	GROPG2G	Does Child Glass object need a relation that is not a DROP type?	Yes	Define DROPG2G relation

This represents the as is state of a current example. I believe that usage of the t_attribute table should be avoided as much as possible. All GROP relationships should rely on t_connector as much as possible. Any use of the DROP patterns should be done carefully. TROP, CROP and PROP patterns are valid. Invariants need to be applied. This is not the final word on usability guidelines.

Boilerplate Columns

EAOM means the manual EA Object Manual.

TABLE NAME	COLUMN NAME	EAOM Class Name	EAOM Class Type	EAOM Read	EAOM Definition
t_attribute	AllowDuplicates	AllowDuplicates	Boolean	RW	Is a duplicate allowed in the collection? If the attribute represents a DB column, then this means NOT NULL.
t_attribute	Classifier	ClassifierID	Long	RW	Local Object ID of the base type for this attribute.
t_attribute	Const	IsConst	Boolean	RW	Const
t_attribute	Container	Container	String	RW	Container
t_attribute	Containment	Containment	String	RW	Containment Type. Values are: Not Specified | By Reference | By Value.
t_attribute	Default	Default	String	RW	Default
t_attribute	Derived	IsDerived	Boolean	RW	Derived
t_attribute	ea_guid	AttributeGUID	String	RO	Global Attribute ID.
t_attribute	GenOption	unexposed			
t_attribute	ID	AttributeID	Long	RO	Local Attribute ID
t_attribute	IsCollection	IsCollection	Boolean	RW	Is the attribute a collection? If the attribute represents a DB column, then this means FOREIGN KEY.
t_attribute	IsOrdered	IsOrdered	Boolean	RW	Is the attribute ordered? If the attribute represents a DB column, then this means PRIMARY KEY.
t_attribute	IsStatic	IsStatic	Boolean	RW	Is the attribute static (immutable)? If the attribute represents a DB column, then this means UNIQUE.
t_attribute	Length	Length	String	RW	Length
t_attribute	LowerBound	LowerBound	String	RW	Collection attribute Lower Bound.
t_attribute	Name	Name	String	RW	Name
t_attribute	Notes	Notes	String	RW	Notes
t_attribute	Object_ID	ParentID	Long	RW	Local Object_ID of Parent
t_attribute	Pos	Pos	Long	RW	Pos
t_attribute	Precision	Precision	String	RW	Precision
t_attribute	Scale	Scale	String	RW	Scale
t_attribute	Scope	Visibility	String	RW	Scope. Values are Public | Private | Protected | Package

TABLE NAME	COLUMN NAME	EAOM Class Name	EAOM Class Type	EAOM Read	EAOM Definition
t_attribute	Stereotype	Stereotype	String	RW	Stereotype
t_attribute	Style	Style	String	RW	Alias optional property.
t_attribute	StyleEx	StyleEx	String	RW	Reserved for Sparx. Do not use.
t_attribute	Type	Type	String	RW	Type
t_attribute	UpperBound	UpperBound	String	RW	Collection attribute Upper Bound.
t_object	Abstract	Abstract	String	RW	If Abstract then 1 else if Concrete then 0. Boolean.
t_object	ActionFlags	ActionFlags	String	RW	Action Semantic Flags.
t_object	Alias	Alias	String	RW	Alias
t_object	Author	Author	String	RW	Author
t_object	Backcolor	unexposed			Use SetAppearance method
t_object	Bordercolor	unexposed			Use SetAppearance method
t_object	BorderStyle	unexposed			Use SetAppearance method
t_object	BorderWidth	unexposed			Use SetAppearance method
t_object	Cardinality	unexposed			
t_object	Classifier	ClassifierID	Long	RW	Local Object_ID of the base type for this object.
t_object	Classifier_guid	unexposed			
t_object	Complexity	Complexity	String	RW	Metric measure. Values are: 1 Easy, 2 Medium, 3 Hard
t_object	Concurrency	unexposed			
t_object	CreatedDate	Created	Date	RW	Created Date
t_object	Diagram_ID	unexposed			
t_object	ea_guid	ElementGUID	String	RO	Global Object ID; valid across models.
t_object	Effort	unexposed			
t_object	EventFlags	EventFlags	String	RW	Signal or event Flags.
t_object	Fontcolor	unexposed			Use SetAppearance method
t_object	GenFile	GenFile	String	RW	Code generation and synchronisation File
t_object	GenLinks	GenLinks	String	RW	Code reversing shows discovered Links to other classes found; Only Parents and Implements connectors.
t_object	GenOption	unexposed			
t_object	GenType	GenType	String	RW	Code generation type. Values are C#, VBNet, Java, etc.
t_object	Header1	Header1	Variant	RW	Code Header to include in a generated source file.
t_object	Header2	Header2	Variant	RW	Code Header to include in a CPP source file.
t_object	IsActive	IsActive	Boolean	RW	Is this an active element?

TABLE NAME	COLUMN NAME	EAOM Class Name	EAOM Class Type	EAOM Read	EAOM Definition
t_object	IsLeaf	IsLeaf	Boolean	RW	Is this a leaf element, so it cannot be a parent element?
t_object	IsRoot	IsRoot	Boolean	RW	Is this a root element, so it cannot have a parent element?
t_object	IsSpec	IsSpec	Boolean	RW	Is this a specification element?
t_object	ModifiedDate	Modified	Date	RW	Modified Date
t_object	Multiplicity	Multiplicity	String	RW	Multiplicity
t_object	Name	Name	String	RW	Name
t_object	Note	Notes	String	RW	Note
t_object	NType	IsComposite	Boolean	RW	Is element composite
t_object	Object_ID	ElementID	Long	RO	Local Object ID; valid only within this model file
t_object	Object_Type	ObjectType	String	RO	Object Type. Type Package | Class
t_object	Package_ID	PackageID	Long	RW	Local Package_ID of parent
t_object	PackageFlags	unexposed			
t_object	ParentID	ParentID	Long	RW	Local Object ID ParentID
t_object	PDATA1	unexposed			
t_object	PDATA2	unexposed			
t_object	PDATA3	unexposed			
t_object	PDATA4	unexposed			
t_object	PDATA5	unexposed			
t_object	Persistence	Persistence	String	RW	Values are: Persistent | Transient.
t_object	Phase	Phase	String	RW	Phase this element will be constructed in.
t_object	RunState	RunState	String	RW	Run State
t_object	Scope	Visibility	String	RW	Scope within package. Values are Public | Private | Protected | Package
t_object	StateFlags	unexposed			
t_object	Status	Status	String	RW	Status
t_object	Stereotype	Stereotype	String	RW	Stereotype
t_object	Style	unexposed			
t_object	StyleEx	StyleEx	String	RW	Reserved for Sparx. Do not use.
t_object	Tagged	unexposed			
t_object	TPos	TreePos	Long	RW	Tree Relative Position; used to sort objects?
t_object	Version	Version	String	RW	Version
t_object	Visibility	Visibility	String	RW	Scope within package. Values are Public | Private | Protected | Package
t_package	BatchLoad	BatchLoad	Long	RW	Is included in Batch Load.

TABLE NAME	COLUMN NAME	EAOM Class Name	EAOM Class Type	EAOM Read	EAOM Definition
					Boolean
t_package	BatchSave	BatchSave	Long	RW	Is included in Batch XMI Export. Boolean.
t_package	CodePath	CodePath	String	RW	Code Source path.
t_package	CreatedDate	Created	Date	RW	Created Date
t_package	ea_guid	PackageGUID	Variant	RO	Global Package ID; valid across models.
t_package	IsControlled	IsControlled	Boolean	RW	Is Controlled
t_package	LastLoadDate	LastLoadDate	Date	RW	XML Last Load Date
t_package	LastSaveDate	LastSaveDate	Date	RW	XML Last Save Date
t_package	LogXML	LogXML	Boolean	RW	Is XMI export logged
t_package	ModifiedDate	Modified	Date	RW	Modified Date
t_package	Name	Name	String	RW	Name
t_package	Namespace	IsNamespace	Boolean	RW	Is package a namespace root?
t_package	Notes	Notes	String	RW	Notes
t_package	Package_ID	PackageID	Long	RO	Local Package ID; valid only within this model file
t_package	PackageFlags	Flags	String	RW	Package Flags
t_package	Parent_ID	ParentID	Long	RW	Tree -if 0 then this package is a model (i.e., no parent)
t_package	PkgOwner	Owner	String	RW	Package Owner
t_package	Protected	IsProtected	Long	RW	Is package marked as Protected?
t_package	TPos	TreePos	Long	RW	Tree Relative Position; used to sort packages.
t_package	UMLVersion	UMLVersion	String	RW	XMI UML export version
t_package	UseDTD	UseDTD	Boolean	RW	Is a DTD used when doing an XMI export?
t_package	Version	Version	String	RW	Version
t_package	XMLPath	XMLPath	String	RW	XML Path to save when using controlled packages.