Article
Iryna Mykhailova · Mar 16, 2023 6m read

Kinds of properties in IRIS

InterSystems IRIS has quite a few different kinds properties. Let ’ s put them in order so that they make better sense.

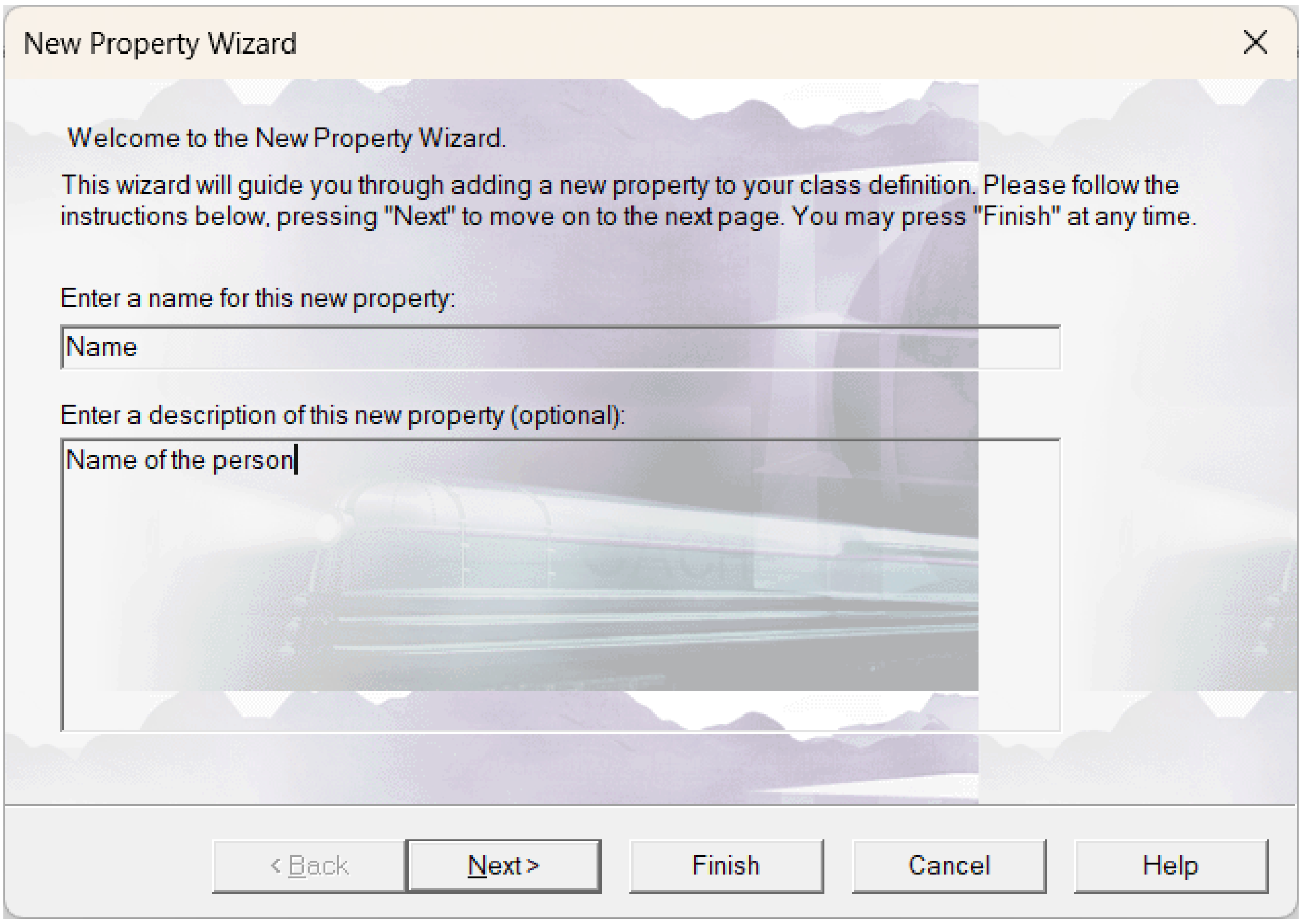
First of all, I would divide them into categories:

- Atomic or simple properties (all those %String, %Integer, %Data and other system or user datatypes)
- References to stored objects
- Built-in objects
- Streams (both binary and character)
- Collections (which are divided into arrays and lists)
- Relationships (one-many and parent-children)

Some of these kinds of properties are quite straightforward. For example, **atomic properties**:

```
Property Name As %Name;
Property DoB As %Date
Property Age As %Integer
```

They are also easily created using Studio Wizard:



New Property Wizard

Welcome to the New Property Wizard.

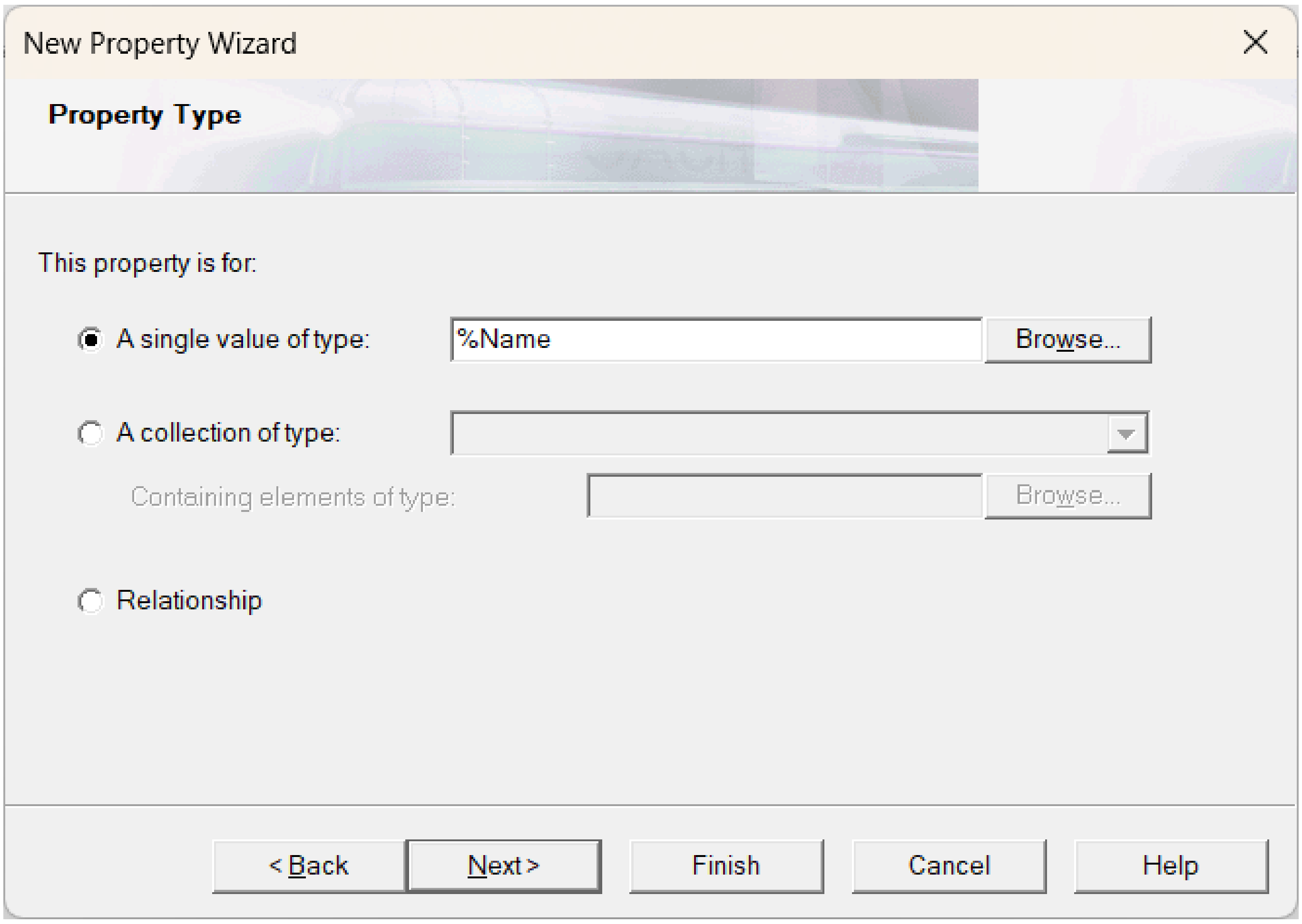
This wizard will guide you through adding a new property to your class definition. Please follow the instructions below, pressing "Next" to move on to the next page. You may press "Finish" at any time.

Enter a name for this new property:

Name

Enter a description of this new property (optional):

Name of the person



New Property Wizard

Property Type

This property is for:

[● A single value of type:] %Name

A collection of type:

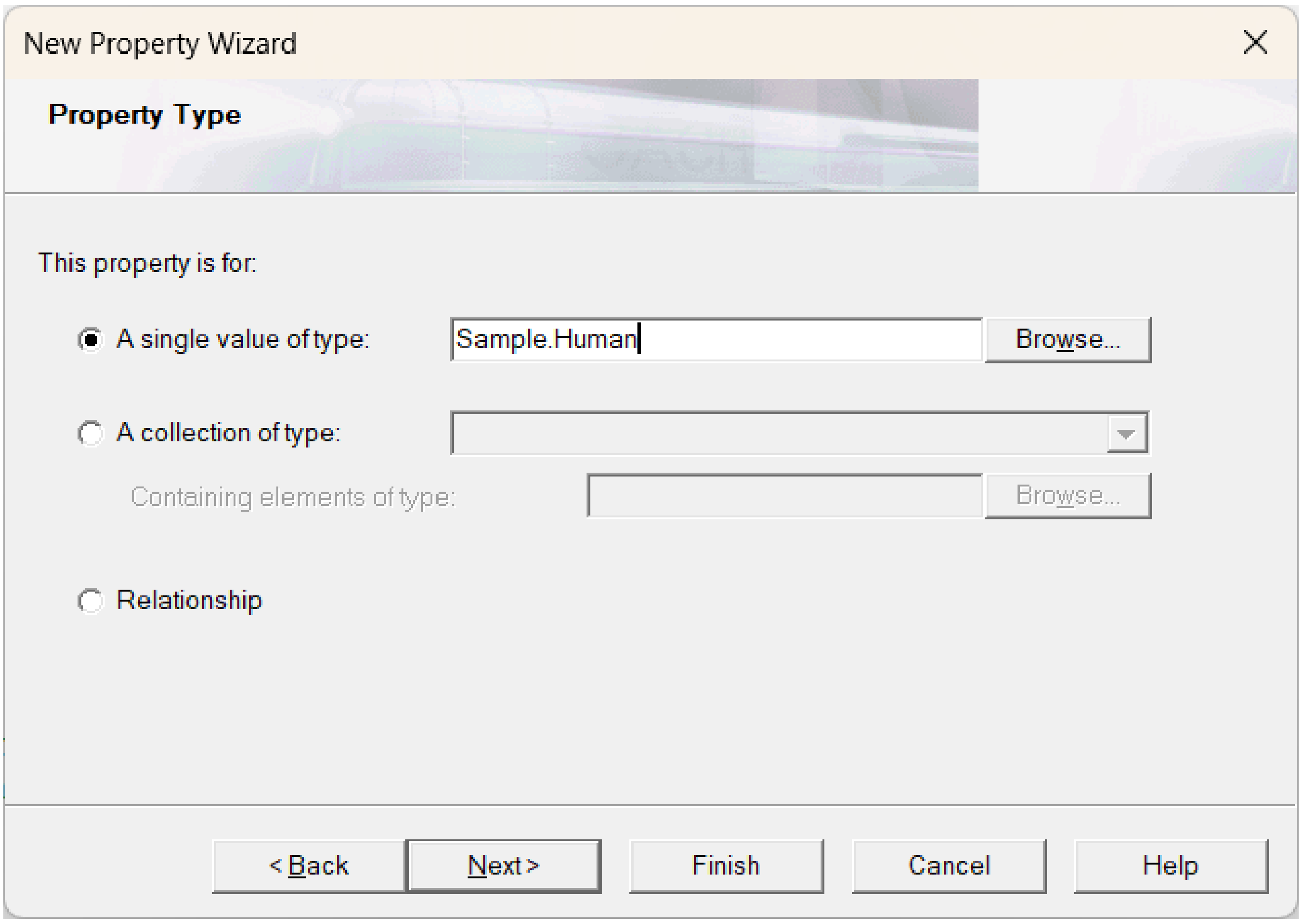
Containing elements of type:

Relationship

The concepts of **references to stored objects** and **built-in objects** are also quite easy to grasp. Basically, if the class of an object you ’ re trying to use as a property extends %Persistent, then it ’ s a reference to a stored object. If the related class extends %SerialObject – then it ’ s a built-in object, because such objects can ’ t be stored themselves, only inside other objects. And in a class definition they look exactly the same:

```
Property Human as Sample.Human;
```

To create this kind of property, on the second step of the Wizard in Studio enter the name of the class:


New Property Wizard

Property Type

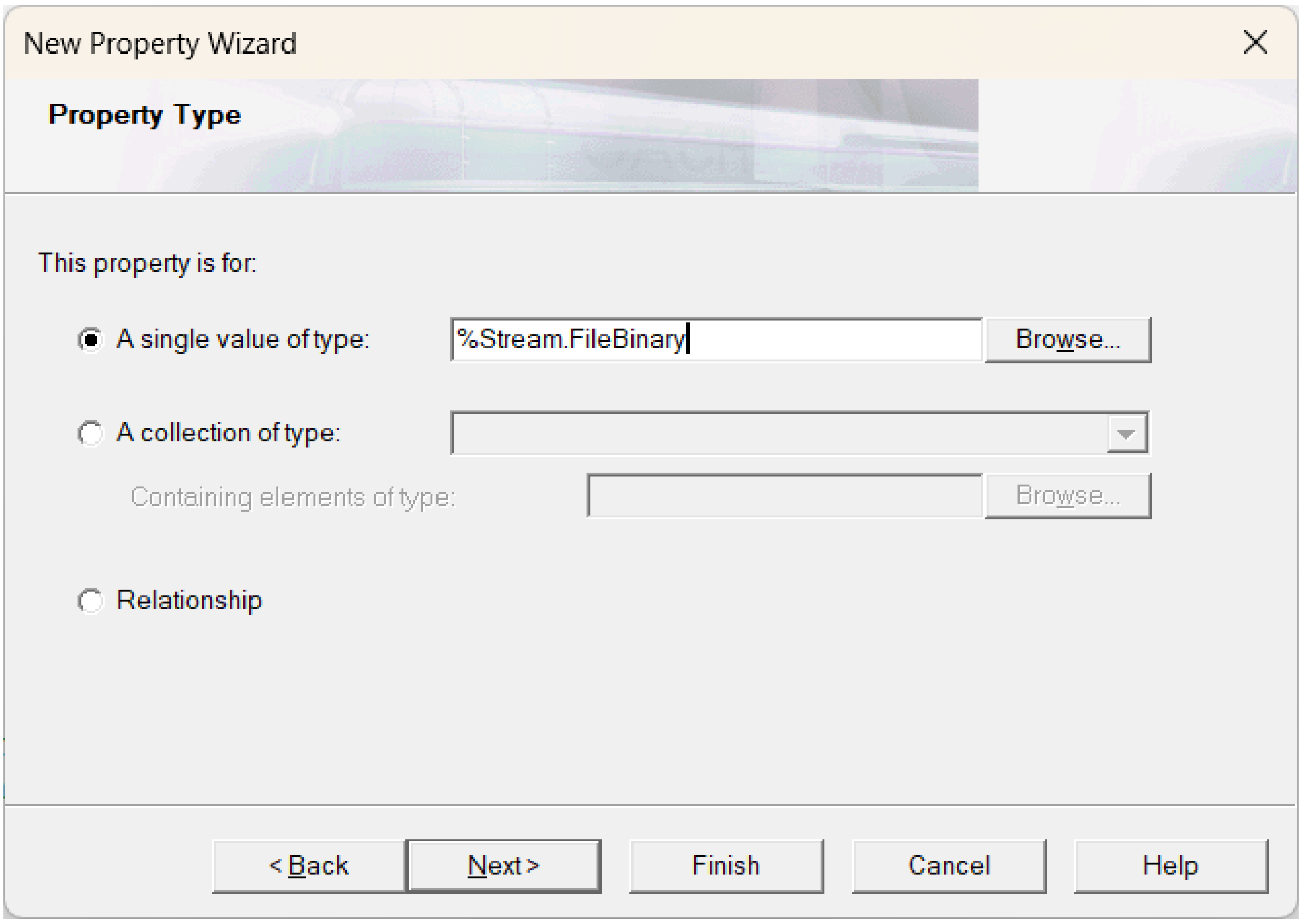
This property is for:

- [✓ A single value of type:] Sample.Human
- A collection of type:
  - Containing elements of type:
- Relationship

Streams are also quite easy to explain. You have a big chunk of unstructured data, be it binary or character, and here is your stream. Can be anything – audio, video, document, text etc. Depending on what type of data you want to store, you may choose to use Binary or Character Stream. Another defining characteristic of a stream is the place where you want to store it – on disk (takes less space but accessible from your OS unless you set up access rules for the folder) or in the database (secure, no unauthorized access, takes more space). To work with streams, use classes from package %Stream. For example:

```
Property Log As %Stream.GlobalCharacter;
Property Instruction As %Stream.FileCharacter(LOCATION = "D:\Temp");
Property Doc As %Stream.FileBinary;
Property Audio As %Stream.GlobalBinary;
```

In this case input the classname of the stream you wish to use:



(●) A single value of type: %Stream.FileBinary

Then there are two types of **collections**:

- Lists – a collection where each element is defined by its position in the collection

```
Property FavoriteColors as list of %String;
```

- Arrays – key-value pairs, where value is defined by its key that is set by a user

```
Property Contacts as array of %String;
```

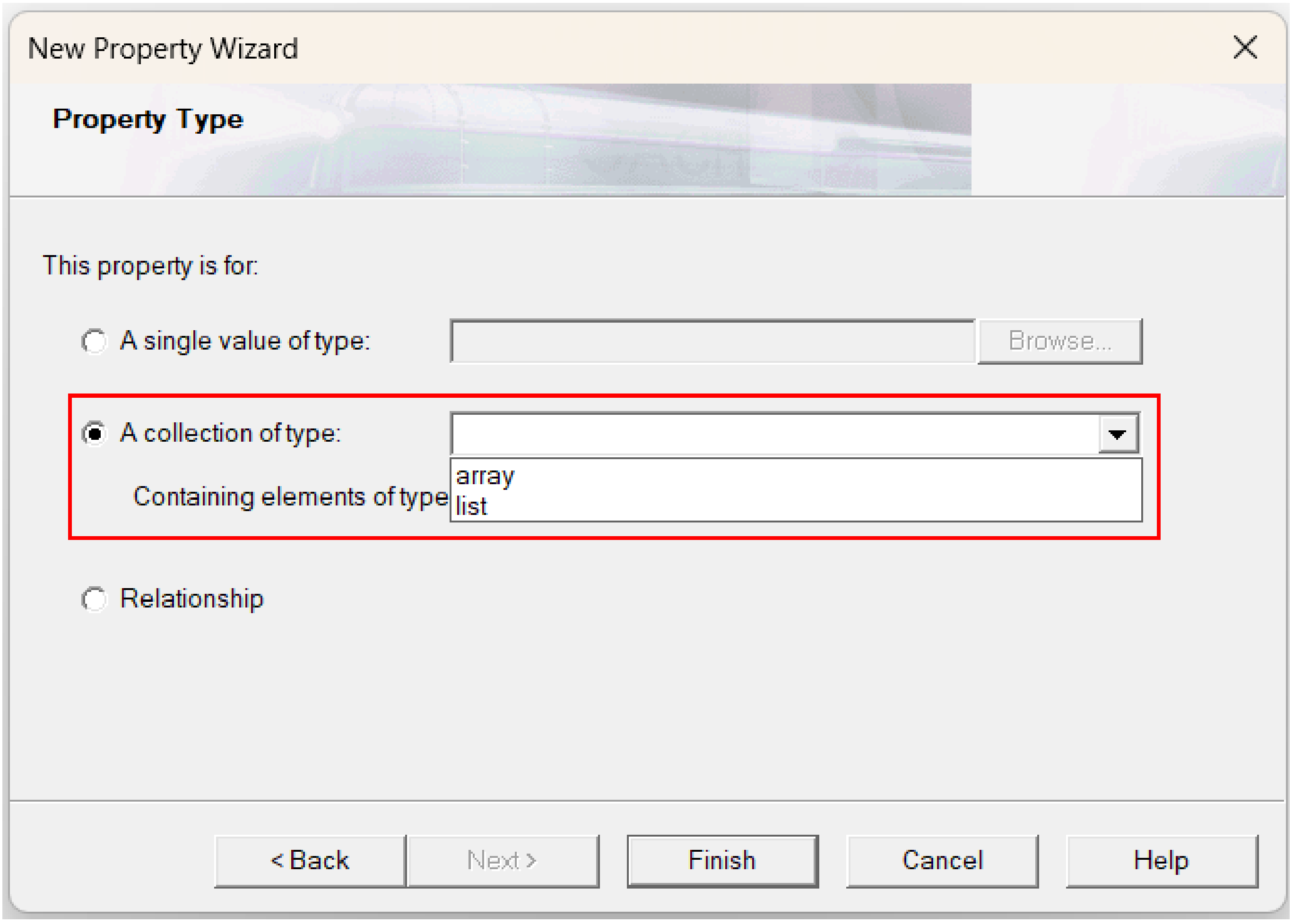
Both collections can be of simple types or of objects. And when we ’ re talking about a collection of objects, the objects inside collection aren ’ t “ aware ” that they are inside any collection. Meaning that it ’ s a one-way link.

When working with arrays, it ’ s necessary to remember that both key and value should be new and useful piece of info. There is no point of making an integer key that will imitate a position in a list. In the example above, key of the Contacts array is a name of the type of the contact, e.g. “ Facebook ” , “ LinkedIn ” , “ Twitter ” etc and the value is a link to a personal page.

Under the hood, when working with a list, you ’ re working with the class %Collection.ListOfDT for a list of simple types, %Collection.ListOfObj when working with a list of objects and %Collection.ListOfStream when dealing with a list of streams.

The same is true for arrays. %Collection.ArrayOfDT is used when working with an array of simple datatypes, %Collection.ArrayOfObj – when working with an array of objects and %Collection.ArrayOfStream – when working with an array of streams.

For the collection on the second step of the Wizard in Studio choose the second option “ The collection of type ”



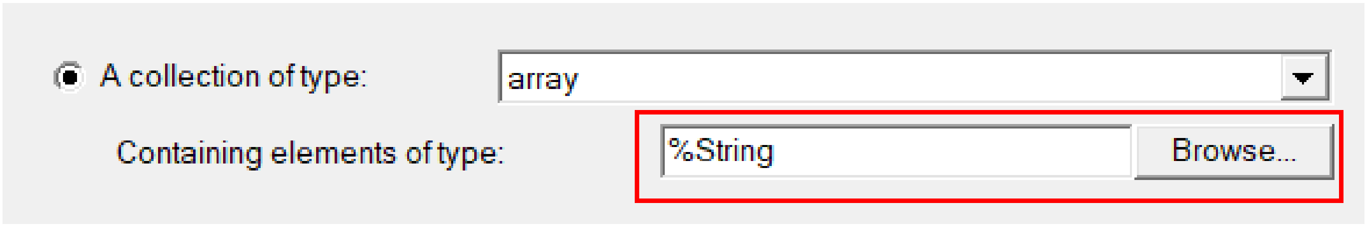
This property is for:

A single value of type:

[✓ A collection of type:]

Containing elements of type

and then specify the type:



[✓ A collection of type:] array

Containing elements of type: %String

And probably the most challenging kind of the property for people who switch from relational databases – **relationship**. The relationship is a two-way one-to-many link between two stored objects. In relational DBMS people are taught that they need an additional table to store the foreign keys to two other tables to realize one-to-many link of independent entities. For example:

table Invoice – table Product – table InvoiceProduct

There is no need for an additional table/class to do this in IRIS. If there ’ s no need to often query the information about all the invoices in which the exact Product is listed you can make products as an array or list in an invoice. In this case you will have to manually ascertain logical integrity so that your invoice doesn ’ t reference products that are no longer in a DB.

To automatically check for the logical integrity of data in this case you can use a relationship between these two classes: Invoice, Product. There are two types of relationships:

- Parent-children – it ’ s a dependent link between objects of two different persistent classes. It means that the dependent object can ’ t exist without the main object. For example, let ’ s say that a chapter in a book can ’ t exist without a book. This will be an example of parent-children relationship, where a book is a main object and chapter is a dependent object and if we delete a book all the chapters from this book will be deleted as well.
- One-many – it ’ s an independent link between objects of one or two persistent classes. It means that both objects can exist and make sense one without the other and if you try to delete the aggregating object, you ’ ll get an error saying that you first have to unlink all the linked objects. If you try to delete the linked objects, they will disappear from the aggregating object. Let ’ s take our invoice and products as an example. We have many products in an invoice. If we try to delete an invoice, we first need to delete all the products from our invoice. If we delete a product, it will disappear from the invoice.

Since it ’ s a two-way link you need to define the properties in both classes. For example in a class Sample.Invoice you will have the definition:

```
Relationship Products As Sample.Product [ Cardinality = many, Inverse = Invoice ];
```

Note that the property is called Relationship and that it has two characteristics:

Cardinality = many – meaning that there are links to many objects inside this property

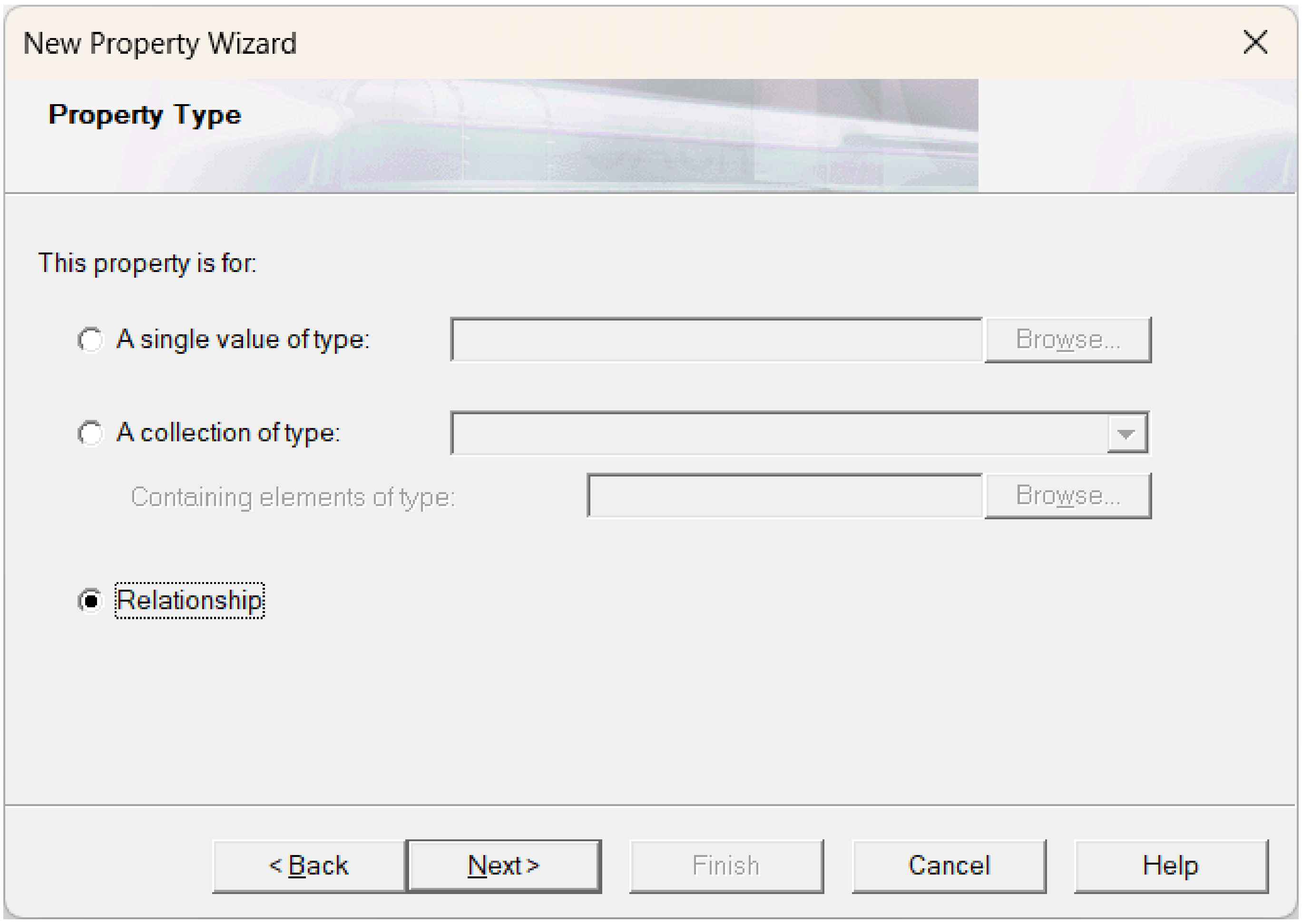
Inverse = Invoice – this is the name of the property on the other side of the relationship

At the same time in the other class (or it can be in the same class for one-many relationship) there should be the mirror property:

```
Relationship Invoice As Sample.Invoice [ Cardinality = one, Inverse = Products ];
```

Here cardinality “ one ” means that in this property there is a link only to one object.

To create a relationship using Wizard in Studio just choose on the second step Relationship:



New Property Wizard

Property Type

This property is for:

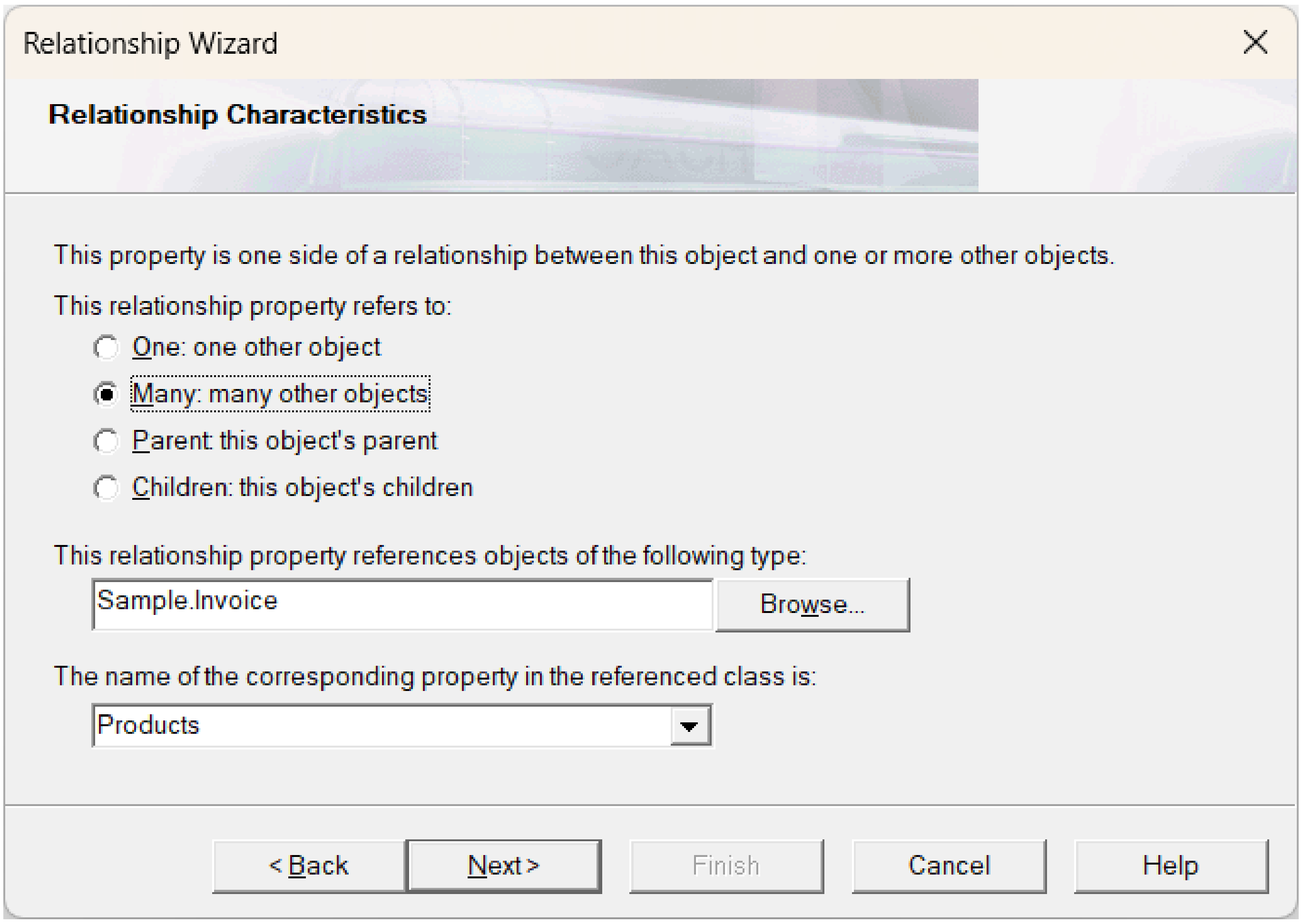
A single value of type:

A collection of type:

Containing elements of type:

[✓ Relationship]

Then choose the correct cardinality for the property in the class and fill in the name of the related class and property in it:


Relationship Wizard

Relationship Characteristics

This property is one side of a relationship between this object and one or more other objects.

This relationship property refers to:

- ( ) One: one other object
- (●) Many: many other objects
- ( ) Parent: this object's parent
- ( ) Children: this object's children

This relationship property references objects of the following type:

Sample.Invoice

The name of the corresponding property in the referenced class is:

Products

For the example with the book with chapters the properties would look as follows.

In a class Sample.Book:

```
Relationship Chapters As Sample.Chapter [ Cardinality = children, Inverse = Book ];
```

In a class Sample.Chapter:

```
Relationship Book As Sample.Book [ Cardinality = parent, Inverse = Chapters ];
```

This is the end of a brief overview of different kinds of properties present in IRIS. Hope it makes it clearer.

#Best Practices #Tutorial #InterSystems IRIS

Source URL:https://community.intersystems.com/post/kinds-properties-iris