Concepts of Data Modeling

Processes are based on data that is going in, is processed, and coming out. This data is specified by data types.
Data types can be structured into packages or interfaces. They are defined by classes and their properties, and have related operations and their parameters.

Icon	Element	Description
	**package**	A package is like a directory for the file system. It is used to group executable data model elements. Packages can have any depth of nesting: To structure your work, you can create packages within packages. Also, packages define a sort of namespace to the contained elements. The name of the package is part of the element path, e.g. **Package1.Class** is different from **Package2.Class**.
	**interface**	In contrast to a class, an interface has no properties nor implementations. Interfaces are used to define common operations of multiple classes, and then derive from that interface. Operations of interfaces do not have an implementation but only define the signature (parameters and types).
	**class**	A class is an aggregation of properties and operations that describes a complex data type from which objects can be created.
	**property**	Properties are data fields that describe the structure of the class.
	**operation**	An operation adds behavior to a class or interface. The behavior describes how to process the data given by the parameters. In the context of the Designer, you can implement operations as mapping, action script or activity.
	parameter	Operations can have parameters that define the input and output objects. Operation parameters can be of simple type (Base Types) or of complex type (class or interface).

Defining Data Types

To describe the data that is used during a process, you can

- use the built-in **Base Types**
- import one or more **libraries** that contain the necessary data types
- create your own **data model (implementation)** with the Designer

Base Types

The Designer provides six base types: **Blob**, **Boolean**, **DateTime**, **Float**, **Integer**, **String** in a standard library that is imported as per default into all service models.
All these types derive from the seventh, general type **Any**.

On this Page:

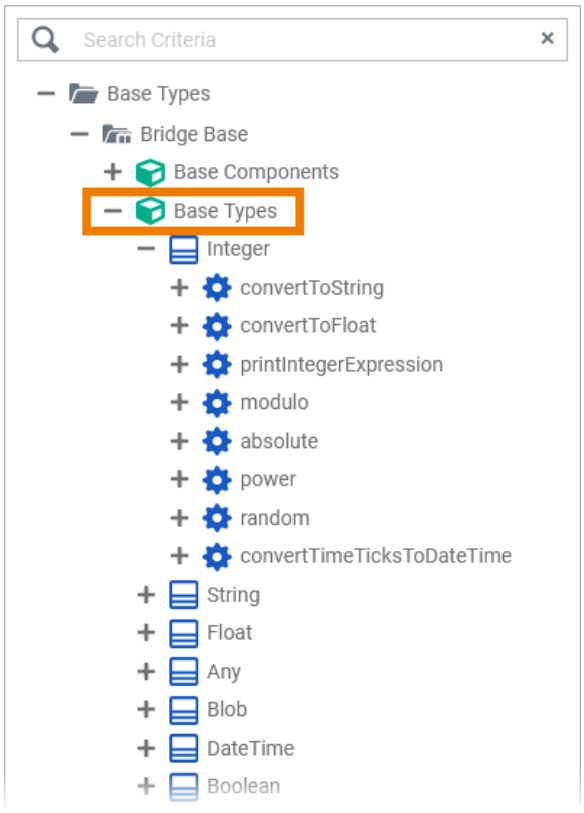
Related Pages:

- Available Base Types
- Custom Complex Types
- XML Serialization

- Mapping Data Structures

Related Documentation:

- PAS Designer User Guide
  - Modeling Data Mapping
  - Modeling Execution
  - Adding Variables
  - Action Script Language
  - Persisting Data
  - Adding Operation Calls
  - Sharing Designer Content

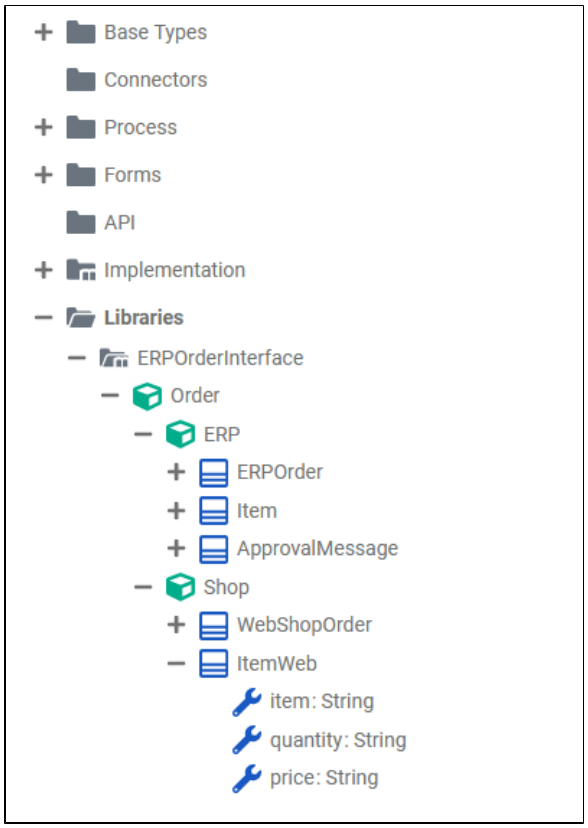


All built-in base types are located in a dedicated package **Base Types** that is provided with the Designer. These base types are only able to hold one single piece of information, like text in a **String**, true or false in a **Boolean**, or binary data in a **Blob**.

You can find more details on the base types and simple examples on Available Base Types.

Libraries

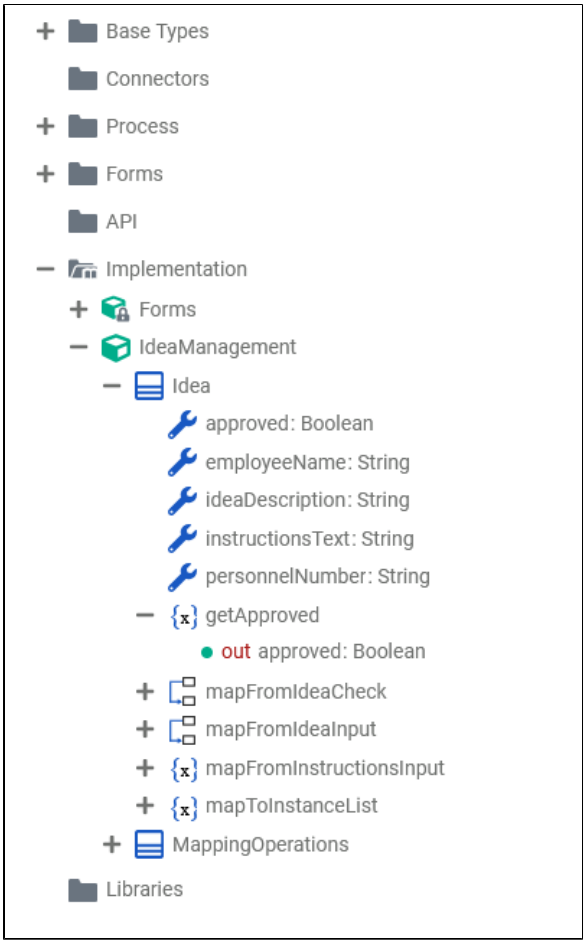
Libraries are code repositories that are useful to organize your development project into re-usable pieces that can be used in multiple services. They contain predefined classes, interfaces, operations and parameters you can use during modeling by simple drag & drop.
Libraries are developed with the Builder. The Designer comes with a standard library which already provides all necessary **Base Types** and base type operations.



The data types provided by a library are listed in the **Service** panel. You can find more details on developing libraries on Sharing Designer Content.

Implementation

Besides the **Base Types** and types from libraries, you can define your own data types in folder **Implementation**.



User defined classes can be structured the same way imported classes are. How to do this is described in detail on Modeling Data Structures.

The **Implementation** folder also contains a locked package **Message**. This package has been generated by the Designer, and holds all classes related to the forms you have created in your service model. You cannot change these generated classes.

Using Data Types

You can use all data types, regardless of where they reside - **Bridge Base**, imported library, or **Implementation**, in your service model in the following places:

- execution diagram
- mapping diagram
- action script

Execution Diagram

Execution diagrams describe what is actually done in a BPMN task. Execution diagrams contain an UML activity flow that can be adorned with incoming and outgoing data items.
Concerning data items, you can

- create and use local variables to store data during this very task
- create persisted variables to store data that should be available in other tasks of the current process
- use variables that have been persisted in a task prior to the present one.

Handling of local and persisted variables is described on Adding Variables and Persisting Data.

Also, data types from imported libraries and from the user-defined **Implementation** may have operations associated that can be used with variables of that type. Data items can go into an operation via parameters, or as a self object.
Refer to

- Adding Operation Calls for more information on how to add an operation call to the activity flow of an execution diagram
- Modeling Data Mapping > Operation for more information on how to create a mapping operation.

You can also create your own class operations in the **Implementation** and implement action script on them. See Action Script below for more details.

Mapping Diagram

Mapping diagrams describe how to transform source data to a target. The types of the source and the target are defined via the input resp. output parameters of the mapping operation.
Refer to

- Modeling Data Mapping for more information on how to create a data model and how to work with the mapping editor
- Mapping Data Structures for more details on the available mapping functions.

Action Script

You can create your own class operations in the **Implementation** and implement action script on them. In a script like fashion, you can use the Action Script Language to implement purposes that are not covered by plain modeling.
You can

- create variables using the create statement.
- use any valid type related operation as listed on Action Script Language pp.

Some basic information on the xUML Action Language regarding the syntax scheme, object references, local variables, NULL values, and constructors are listed on Basics of the Action Script Language.