

# Project Organization

When starting with a new project, you should give some thoughts to the project organization. To set up a change-friendly, clear project organization, you should take the following questions into account:

1. [Builder project - when to use a single Builder project, when to split into multiple projects?](#)
2. [Service models - where to log changes?](#)
3. [Service model - how to organize external specifications, such as shared data structures, libraries and modules?](#)

## When to Use a Single Builder Project, When to Split into Multiple

Splitting-up the project should base on logical conditions, but not on project team organization. This means: Do not create one Builder project for every team member, but e.g. one for every customer organization unit. The count of models in one Builder project is not limited technically, but more by logical reasons. At a certain amount of models in a project, it will be difficult to stay in top of things.

Additionally we recommend to collect all libraries and modules in one Builder project. These libraries and modules then can easily be reused in other Builder projects. The initial work on these libraries and modules still can be done in a specific Build project. If the library/module is ready, move it to the libraries and modules project.

## When to use Libraries, Modules or (Base) Services

This is actually an architectural decision depending on the architecture of the installation site (e.g. zone concept). In general, Base Services in a SOA context can be web services or libraries

Implementation Type	Use, when
Base Services	<ul style="list-style-type: none"><li>• the service implements a communication between nodes or zones</li><li>• performance is not a big issue</li><li>• a third party wants to use the functionality, too, via web service</li><li>• complex interfaces need to be simplified</li></ul> <p><b>Advantage:</b> If the backend interface changes, Base Service interface does not necessarily need to be changed.</p> <p><b>Drawback:</b> If the Base Service Interface changes, all consumers must change, too.</p>
Module	<ul style="list-style-type: none"><li>• you need the same implementation in several xUML services with different name and ports</li><li>• you just want to share data structures or packages and no interfaces (operations) are required</li></ul>
Library	<ul style="list-style-type: none"><li>• you want to encapsulate an implementation/special functionality of an operation</li><li>• you want to encapsulate backend functionality (only possible, if it is not used by a third party, means: no frontend (e.g. SOAP))</li></ul> <p><b>Advantages:</b></p> <ul style="list-style-type: none"><li>• Better version control through logic version within library model (component diagram)</li><li>• Better control, when a changed library is deployed within a service</li></ul>

## When to Set a New Logical Library Version Number

The library version number can be set in the component diagram. Change the library version number after implementing a new requirement, that will be deployed and don't forget to document the change as described in [Where to Log Changes](#) further below.

Use major and minor versions:

- Change the major version, if the interface has changed,.
- Change the minor version for each implemented new requirement.

### On this Page:

- [When to Use a Single Builder Project, When to Split into Multiple](#)
  - [When to use Libraries, Modules or \(Base\) Services](#)
  - [When to Set a New Logical Library Version Number](#)
  - [Service Oriented Architecture](#)
- [Where to Log Changes](#)
  - [Using a Version Control Tool](#)
    - [Which Files to Put Under Version Control](#)
  - [Port Organization / Service Inventory](#)
- [How to Organize Shared Data Structures, Libraries and Modules](#)

### Related Pages:

- [Project Organization](#)
- [Naming Conventions and Containment Tree Organisation](#)
- [Model Documentation](#)
- [Settings](#)
- [Mappings](#)
- [Sub-activities](#)
- [Logging](#)
- [Error Handling](#)

## Service Oriented Architecture

Split your service into base services or libraries (also see [When to use Libraries, Modules or \(Base\) Services](#) above), e.g. to abstract backends.

This is useful for:

- Zone transfer (infrastructure for communication between secure zone and dmz zone, e.g. SOAP over https)
- Concerning libraries: Changes have to be compiled into (base) services only if the signature changes (make generic interfaces).

Examples:

- Salesforce-Base service
- database abstraction
- FTP service (dispatcher)

## Where to Log Changes

The truth of what has been changed is only in the model!

So, we recommend to track changes in a change log (e.g. in a note) on the use case diagram of the service model. This especially helps, if the source is located in multiple places, e.g. at the customer site and locally at the site of the project team.

Such a change log should contain at least the following information:

- the date
- the author of the edit
- the model version
- what has been done
- We also recommended to add a comment, if a new version of a library is used.

The Builder templates provide an default change log on the use case diagram that you simply can fill in.

Copy the lines from the change log and add them also to the check-in comment of your versioning system.

## Using a Version Control Tool

Using a version control tool to control changes of models needs some discipline. With SVN, the following approach has been proved useful:

1. Get the current version of the model.
2. Lock the model for concurrent edits.
3. Check in when work is done.

When two or more team members are working on the same service, it may be helpful to split the service into modules that can be worked on independently (e.g. mapping module, ...).

## Which Files to Put Under Version Control

You should control the following files by a version control system:

- all files from following folders:
  - **uml** and **imports**
  - **libs**
  - **jarfiles**
  - **resource**
- the **.e2ebuilder** file, but not **.\$e2ebuilder.workspace** (which contains temporary data like "last project saved" and so on)
- concerning testing:
  - Create a separate Builder project that contains your tests, especially for regression tests, and put that under version control (files and folders see below)
  - folder **testcase**
  - Best is, to introduce a QA server with the Analyzer only installed.

If you need to rename or remove a folder that is under version control, you can

- zip the old folder and check in the ZIP file.

- remove the old ones from your version control system and check in the new (if applicable)

Depending on your version control system, you will probably lose history then.

## Port Organization / Service Inventory

We recommend to keep a service inventory in a central space, e.g. Confluence or any other wiki, operation manual (Word) or Excel list.

Make an inventory of:

- ports (control and service ports)
- service names
- service category (for E2E Bridge)
- short description (overview)
- model names

## How to Organize Shared Data Structures, Libraries and Modules

Collect all specification sources (e.g. XSD, DDL, WSDL, CSV) in particular sub-folders of a folder **specifications**. Locate this folder inside the Builder project, if the specifications are specific to this project - locate it outside the Builder projects, if the specifications are shared by multiple projects.

After being imported to a Builder project, these sources will reside in the **uml/imports** folder of this specific Builder project. If sources are used in multiple Builder projects, you have to keep the import folders redundantly for each project.

You should document dependencies of imported elements in the descriptive use case diagram.

Element	Remark
<b>Library</b>	<ul style="list-style-type: none"> <li>• The source of the library should reside in an own Builder project.</li> <li>• You can also use the library dependency dialog from the <b>Tools</b> menu of the E2E Compiler to get an overview on library dependencies.</li> </ul>
<b>Module</b>	<ul style="list-style-type: none"> <li>• The source of the module resides either in an own Builder project (if the module can be used in multiple projects), or in the Builder project the module has been designed for.</li> <li>• Use revision tag from your version control to distinct the module versions.</li> </ul>