3 Basic elements of Papyrus Objects

The Papyrus frameworks and the Papyrus applications are based on the object-oriented Papyrus Objects application development system. Frameworks are composites of templates and instances derived from Papyrus Objects classes and templates composites are blueprints of Papyrus applications. The Papyrus Objects application development system consists of a number of basic elements.

3.1 *Main components of Papyrus Objects*

Papyrus Objects consists of the following main components.

Object

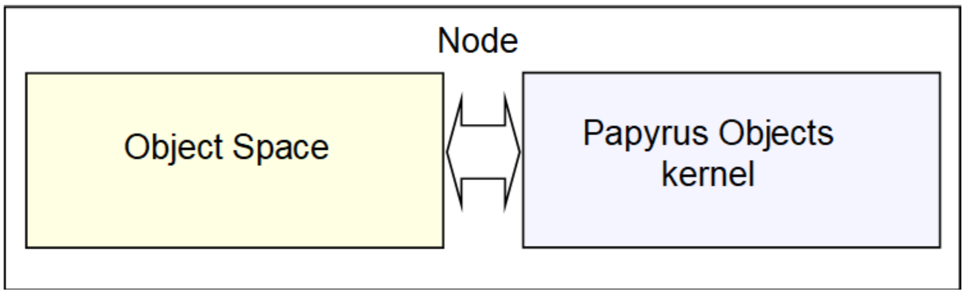
An **object** is the smallest functional unit in the Object Space. It is a data structure which consists of attributes, methods (operations, functions) that can be applied to it or to derived data structures, states and their state transitions, describing the life-cycle of the object. An object can have parent and child references.

Object Space

The **Object Space** is a database that stores all objects of a node of a Papyrus Objects domain.

Node

A **node** is an organizational unit of the Papyrus Objects domain consisting of an Object Space to store the objects and the Papyrus Objects kernel running it. The node is connected to the rest of the Papyrus Objects domain with the Domain Controller provided by Papyrus Software as the main node handling user access and providing classes, templates, and resources.

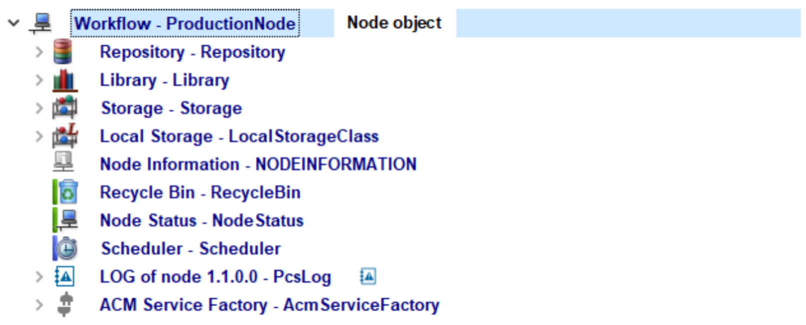


Object Space and kernel make a node

A node runs on a machine which is connected by TCP/IP and UDP to the Papyrus Objects domain (LAN or WAN). On one physical machine there may be more than one node running simultaneously. The graphical user interface Papyrus Desktop is optional and is automatically installed if available for the operating system specified in the installation routine. Alternatively, it is also possible to use a web browser if Papyrus WebPortal is installed on the node whose Object Space should be accessed (thin client solution).

Node object

The **Node object** is the root element of the Object Space and represents the node (Object Space + Papyrus Objects kernel).



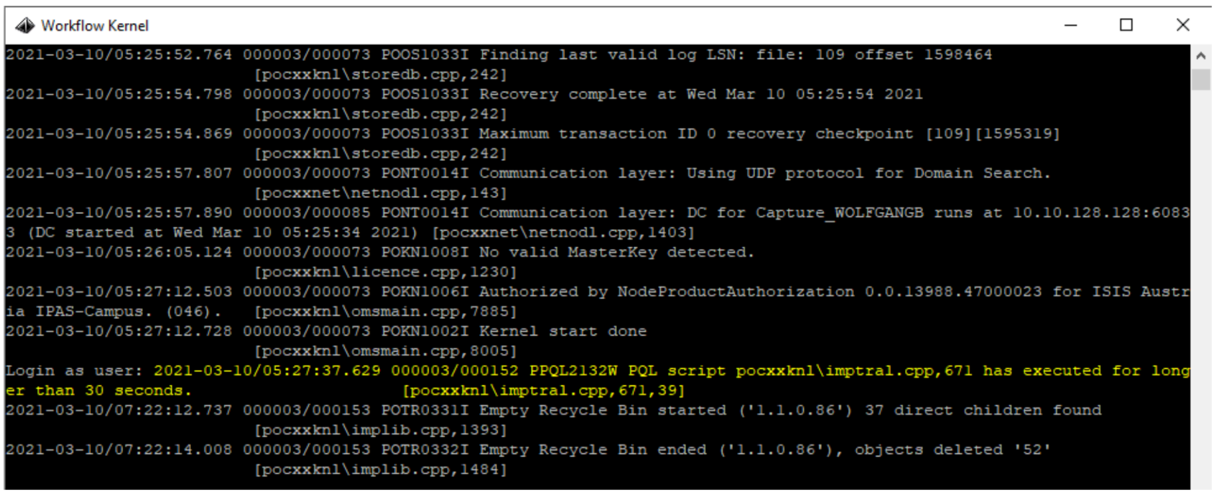
Workflow - ProductionNode   Node object
Repository - Repository
Library - Library
Storage - Storage
Local Storage - LocalStorageClass
Node Information - NODEINFORMATION
Recycle Bin - RecycleBin
Node Status - NodeStatus
Scheduler - Scheduler
LOG of node 1.1.0.0 - PcsLog
ACM Service Factory - AcmServiceFactory

Components of a Node object

Papyrus Objects kernel

The **Papyrus Objects kernel** is the core system managing the Object Space and the communication between the nodes of a Papyrus Objects domain.

The Papyrus Objects kernel is represented by the kernel window of a particular node.



Workflow Kernel

```
2021-03-10/05:25:52.764 000003/000073 POOS1033I Finding last valid log LSN: file: 109 offset 1598464
                         [pocxxknl\storedb.cpp,242]
2021-03-10/05:25:54.798 000003/000073 POOS1033I Recovery complete at Wed Mar 10 05:25:54 2021
                         [pocxxknl\storedb.cpp,242]
2021-03-10/05:25:54.869 000003/000073 POOS1033I Maximum transaction ID 0 recovery checkpoint [109][1595319]
                         [pocxxknl\storedb.cpp,242]
2021-03-10/05:25:57.807 000003/000073 PONT0014I Communication layer: Using UDP protocol for Domain Search.
                         [pocxxnet\netnodl.cpp,143]
2021-03-10/05:25:57.890 000003/000085 PONT0014I Communication layer: DC for Capture_WOLFGANGB runs at 10.10.128.128:6083
3 (DC started at Wed Mar 10 05:25:34 2021) [pocxxnet\netnodl.cpp,1403]
2021-03-10/05:26:05.124 000003/000073 POKN1008I No valid MasterKey detected.
                         [pocxxknl\licence.cpp,1230]
2021-03-10/05:27:12.503 000003/000073 POKN1006I Authorized by NodeProductAuthorization 0.0.13988.47000023 for ISIS Austr
ia IPAS-Campus. (046).   [pocxxknl\omsmain.cpp,7885]
2021-03-10/05:27:12.728 000003/000073 POKN1002I Kernel start done
                         [pocxxknl\omsmain.cpp,8005]
Login as user: 2021-03-10/05:27:37.629 000003/000152 PPQL2132W PQL script pocxxknl\imptral.cpp,671 has executed for long
er than 30 seconds.                     [pocxxknl\imptral.cpp,671,39]
2021-03-10/07:22:12.737 000003/000153 POTR0331I Empty Recycle Bin started ('1.1.0.86') 37 direct children found
                         [pocxxknl\implib.cpp,1393]
2021-03-10/07:22:14.008 000003/000153 POTR0332I Empty Recycle Bin ended ('1.1.0.86'), objects deleted '52'
                         [pocxxknl\implib.cpp,1484]
```

Kernel window

Object Space components

The objects in an Object Space of a node of a Papyrus Objects domain are organized in a hierarchical structure.

The Node object is the root object of the object tree of an Object Space. All other objects of the Object Space are part of the node object tree.

The following objects are the main components of an Object Space:

Node	In Papyrus Objects, a **Node** object is the root of that node's Object Space. All node objects of a Papyrus Objects domain reference the same Repository, Library and Storage components as children. Each of these components can likewise be expanded when displayed in tree view.
Repository	The **Repository** contains **class** objects. These are hierarchically arranged, with base classes displayed as parents and subclasses displayed as children. The "branches" of each subtree can independently be sorted on visible name, internal name or time of creation.
Library	The **Library** contains **template** trees, which are organized into library folders. Each template tree represents a collection of objects that can be instantiated.
Storage	The **Storage** holds **collections** of system-relevant objects that are used across the domain. Objects that are used only by specific nodes, users or applications should not be placed in Storage.

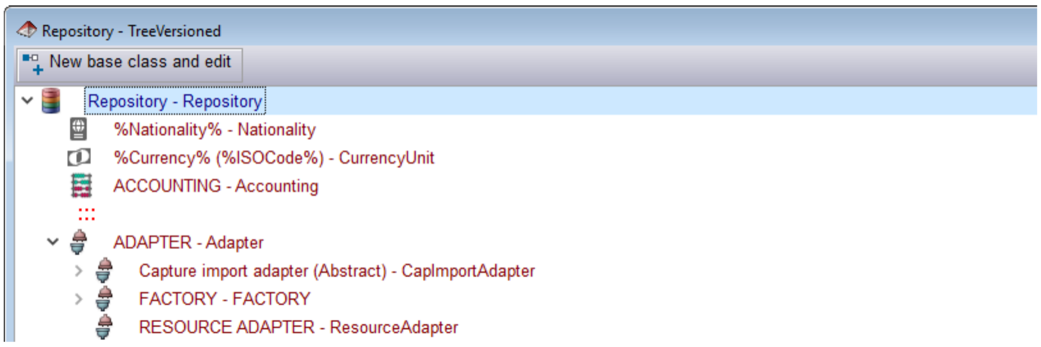
Object types

Objects in an Object Space can be distinguished in three different types with respect to their position in the development cycle: classes, templates and instances.

Classes

Classes have red labels on the Papyrus Desktop and reside exclusively in the **Repository.**

Classes are **definitions of objects and their behavior**. Classes define the general properties and behavior of objects derived from these classes. Developers design classes in terms of their members, with attributes defining the content, methods defining the actions, state machines defining states and their transitions and reference definitions defining parent and child relations.



Repository - TreeVersioned

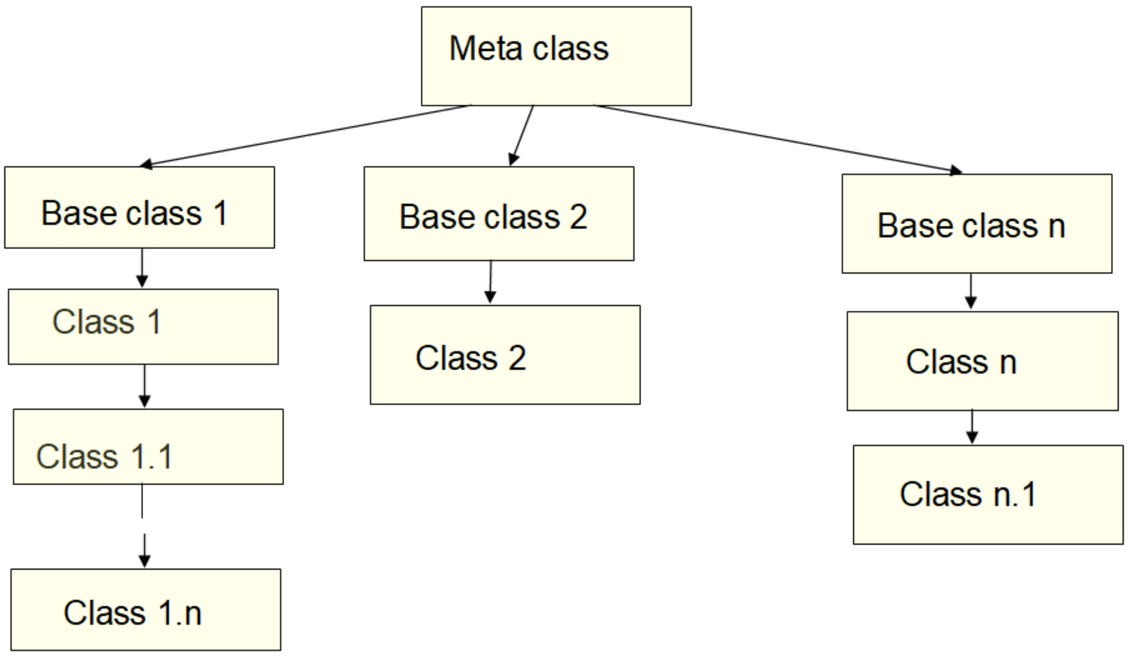
New base class and edit

- Repository - Repository
  - %Nationality% - Nationality
  - %Currency% (%ISOCode%) - CurrencyUnit
  - ACCOUNTING - Accounting
  - ADAPTER - Adapter
    - Capture import adapter (Abstract) - CapImportAdapter
    - FACTORY - FACTORY
    - RESOURCE ADAPTER - ResourceAdapter

Repository

The object tree in the Repository reflects the inheritance chain. A child class (=derived class) inherits all properties (methods, attributes, states, state transitions, and parent/child reference definitions) of its parent class.

Direct child objects of the **Repository** are called **base classes**. The objects below a base class are derived from it and are called child classes or derived classes. Child classes can be defined recursively. The base class of all classes in the Repository is the class **MetaClass** which defines features required by all child classes (e.g. methods for instantiation, referencing, duplication, changing policies, etc.).

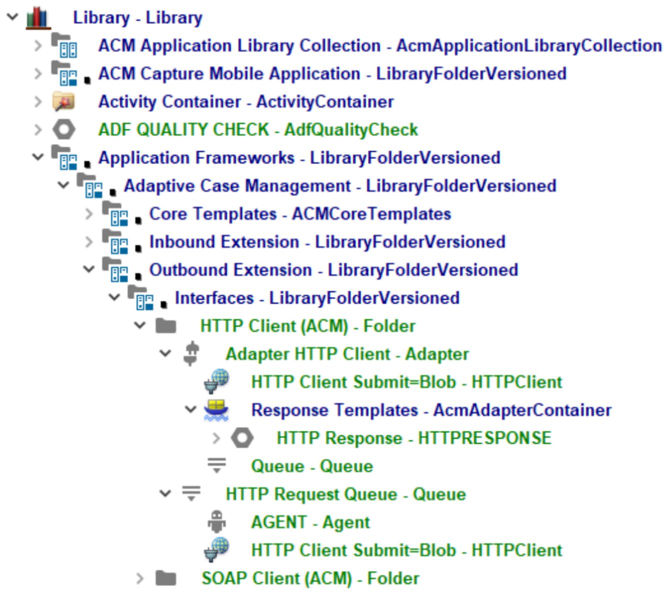


Inheritance of classes in the Repository

The figure above shows the inheritance relations between classes in the the Repository. Children of the Meta class are base classes. Each base class can have a derived class as child and each derived class can have derived child classes recursively. Each class in the tree inherits the features of its parent class recursively.

Templates

Templates have green labels on the Papyrus Desktop and reside in the **Library**. Within a **general container**, however, it is possible to put templates anywhere. This is because adapters have general containers to instantiate their task templates.



```
Library - Library
  ACM Application Library Collection - AcmApplicationLibraryCollection
  ACM Capture Mobile Application - LibraryFolderVersioned
  Activity Container - ActivityContainer
  ADF QUALITY CHECK - AdfQualityCheck
  Application Frameworks - LibraryFolderVersioned
    Adaptive Case Management - LibraryFolderVersioned
      Core Templates - ACMCoreTemplates
      Inbound Extension - LibraryFolderVersioned
      Outbound Extension - LibraryFolderVersioned
        Interfaces - LibraryFolderVersioned
          HTTP Client (ACM) - Folder
            Adapter HTTP Client - Adapter
              HTTP Client Submit=Blob - HTTPClient
              Response Templates - AcmAdapterContainer
                HTTP Response - HTTPRESPONSE
              Queue - Queue
            HTTP Request Queue - Queue
              AGENT - Agent
              HTTP Client Submit=Blob - HTTPClient
          SOAP Client (ACM) - Folder
```

Example of a ready-to-use template setup

Templates are **production descriptions** that represent **specific applications** and are derived from the generally defined classes.

Administrators derive templates from classes, combine them into tree structures and configure them assigning default values to their attributes. The template tree structures are modelling applications for a particular purpose (e.g. an adapter for data import).

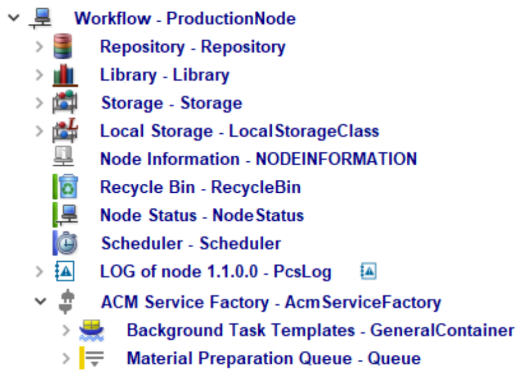
The object tree of templates or instances does not reflect an inheritance chain, its objects are just combined in a functional relationship.

Unlike in the Repository, the parent-child relation between objects of a template tree does not imply inheritance. Instead, these parent-child relations are functional only.

Instances

Instances have blue labels, are derived from **templates** or **classes**, and represent real objects.

Instances typically reside below the node object but they can be anywhere except in the **Repository**.


Workflow - ProductionNode
Repository - Repository
Library - Library
Storage - Storage
Local Storage - LocalStorageClass
Node Information - NODEINFORMATION
Recycle Bin - RecycleBin
Node Status - NodeStatus
Scheduler - Scheduler
LOG of node 1.1.0.0 - PcsLog
ACM Service Factory - AcmServiceFactory
Background Task Templates - GeneralContainer
Material Preparation Queue - Queue

Instances as children of a node object

Instances are objects that are used during production. Whereas classes and templates serve to define instances, instances themselves contain real data and assume actual states. They can be derived (i.e. "instantiated") from classes as well as class and composite templates. Like templates, instances can be referenced and duplicated.

Unversioned Instances

Unversioned instances store data and configurations that are intended for one-time usage.

Versioned Instances

Versioned instances are intended for reusable content that might change over time (e.g. resource objects, rule objects with scripts to execute, etc.). Versioning enables you to duplicate the resource and to change it without losing its prior versions. Future versions can be prepared in advance that become valid at a certain date and time. You can even simulate an earlier date by generating a document with resource versions that were valid in the past.

Framework

Auto and specific

A **framework** is a set of classes, templates, optionally instances and references to other frameworks that combine objects together to construct an abstract design of a specific business application. Business Correspondence, Customer Response Management, Adaptive Case Management and Business Document Capture are examples of business applications.

The customer can make use of the frameworks and develop solutions to meet their business requirements.