1. Certification Study Notes - Core

[1. Certification Study Notes - Core 1](#_Toc442793059)

[Introduction 3](#_Toc442793060)

[Spring, OO, Java 3](#_Toc442793061)

[Platform Basics 5](#_Toc442793062)

[Full installation 8](#_Toc442793063)

[Directory Structure 8](#_Toc442793064)

[Administrating and Running hybris 8](#_Toc442793065)

[Removing the hybris Server Service 8](#_Toc442793066)

[Build Framework(3) 9](#_Toc442793067)

[Type System 12](#_Toc442793068)

[Enumerations 14](#_Toc442793069)

[Type Deployment 15](#_Toc442793070)

[Architecture 16](#_Toc442793071)

[Extensions(7) 21](#_Toc442793072)

[MCC 24](#_Toc442793073)

[hMC 25](#_Toc442793074)

[ImpEx(3) 26](#_Toc442793075)

[Cronjobs (2) 28](#_Toc442793076)

[Flexible Search (5) 29](#_Toc442793077)

[Beans and Enum Generation 30](#_Toc442793078)

[Dynamic Attributes(2) 30](#_Toc442793079)

[Service Layer(4) 31](#_Toc442793080)

[Interceptors 34](#_Toc442793081)

[Workflow 34](#_Toc442793082)

[Tasks 35](#_Toc442793083)

[Classification 36](#_Toc442793084)

[Cockpit NG 37](#_Toc442793085)

[Web Services 38](#_Toc442793086)

[Third Party Databases 39](#_Toc442793087)

[MySQL 39](#_Toc442793088)

[Security / Restrictions 40](#_Toc442793089)

[Transactions 40](#_Toc442793090)

[Caching 41](#_Toc442793091)

[Validation 41](#_Toc442793092)

| **Topic** | **No. questions** |
| --- | --- |
| Creating extensions in hybris | 7 |
| Administrating and Running hybris | 6 |
| Flexible Search | 5 |
| Service layer | 4 |
| Impex | 3 |
| Building hybris project | 3 |
| Cockpit NG | 3 |
| Caching | 3 |
| ORM in hybris | 2 |
| OO & Architecture in hybris | *2* |
| Cron Jobs | 2 |
| Dynamic attributes | 2 |
| Spring Programming | 2 |
| PCM Basics | 2 |
| Security | 1 |
| Cockpits | 1 |
| Web Service API | 1 |
| Testing | 1 |

# Introduction

* Pages marked with the 'certification\_hint' on the wiki: [/label/release5/certification\_hint?startIndex=0](https://wiki.hybris.com/label/release5/certification_hint?startIndex=0)
* <https://wiki.hybris.com/display/pe/The+hybris+Developer+Story+-+hybris+5.X>

# Spring, OO, Java

* Two hybris custom Spring scopes: tenant (deprecated)  and yrequest (like normal request scope, but also outside web app ctx).
* One global app ctx with one ctx per tenant as children. Under each tenant are web app contexts.
* Three xml files for bean definitions
  + **global-{ext-name}-spring.xml** (beans are shared among all extensions, global context)
  + **{ext-name}-spring.xml** (beans shared among all extensions, and will have as many instances as there are tenants, tenant context)
  + **web-application-config.xml** (beans available for selected extension and per tenant, web app context)
* Extend global in **<extname>-global-spring.xml**, put file in **<extname>/resources**.
* Add line to project.properties (<**extname>.global-application-context=<extname>-global-spring.xml**).
* Use **<extname>.application-context=**(comma separated list) to use more than one config file.
* Get an instance of global context with **Registry.getSingletonGlobalApplicationContext**.
* Extend core **<extname>-application-context.xml** (or own filename in **<extname>/resources** and specify it in project.properties, eg. **<extname>.application-context = <extname>-spring.xml**).
* To control extension build and ctx load time, use "required" in extensioninfo.xml (**<requires-extension>**).
* **Registry.getCoreApplicationContext()** gives Core app context.
* To have Core ctx as parent to a web app ctx, initialize with HybrisContextLoaderListener.
* Use **Registry.getApplicationContext**.
* Also set a RequestContextListener in web.xml, to scope beans to request or session.
* MVC DispatcherServlet introduces further sub-context to web app ctx.
* Use init-param element on DispatcherServlet, to specify ctxCfgLocation (in root level of web.xml, the web app ctx cfg is specified).
* [Implementation Principles for Services](https://wiki.hybris.com/display/release5/Implementation+Principles+for+Services)
* [Spring Framework in the hybris Commerce Suite](https://wiki.hybris.com/display/release5/Spring+Framework+in+the+hybris+Commerce+Suite)
* [Spring ApplicationContexts in hybris](evernote://wiki.hybris.com/view/14342068/s124/71df955b-59a8-449e-bd69-a67ca2c3f38b/71df955b-59a8-449e-bd69-a67ca2c3f38b/)

### Bean Scopes

Apart from regular bean scopes, hybris Commerce Suite features two special scopes: **yrequest** and the deprecated **tenant** scope.

Apart from regular bean scopes, hybris Commerce Suite features two special scopes: **yrequest** and the deprecated **tenant** scope.

#### yrequest Scope

The **global-core-spring.xml** file of the **core** extension adds a special scope named **yrequest** to the core **ApplicationContext**. The **yrequest** scope may be used for binding beans to the application context for a single request. Upon session deactivation, the beans are no longer referenced from the application context, allowing them to be garbage collected. This is similar to a standard Spring **request** scope but is used outside of a web application context.

As an example use case, assume you have a third party invoice processing server that requires pulling order data from a hybris system at periodic intervals. Instead of using web services or some other means of data transfer, you decide to use the hybris API directly from within the invoice processing server. You use the **yrequest** scope in this case, so that all beans bound for a request from the invoice server are released when the request completes.

An example of how to set a bean's scope to **yrequest**:

|  |
| --- |
| <bean id="myExtension.myBean" class="de.hybris.platform.myExtension.myClass" scope="yrequest" /> |

#### Tenant Scope

Since version 5.0 of hybris Commerce Suite, tenant scope is no longer in use:

1. If you use xml-based annotations, remove the following reference from your XML configuration files:

scope="tenant"

1. If you use @Scope("tenant") annotations in your java classes, add an ignore-scope-resolver to your XML configuration files:

|  |
| --- |
| <context:component-scan base-package="de.hybris.platform.b2b"  scope-resolver="de.hybris.platform.spring.IgnoreTenantScopeMetadataResolver" |

1. In web configuration, if any of the files: **WEB-INF/xxxstore-web-spring.xml** , **WEB-INF/springmvc-servlet.xml** , **WEB-INF/security.xml** contains a tenant scope usage, you may encounter Web application startup problems. The solution is to use **TenantIgnoreXmlWebApplicationContext**

|  |
| --- |
| <servlet>          <servlet-name>springmvc</servlet-name>          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>          <init-param>              <param-name>contextClass</param-name>              <param-value>de.hybris.platform.spring.ctx.TenantIgnoreXmlWebApplicationContext</param-value>          </init-param>          <load-on-startup>1</load-on-startup>      </servlet> |

1. **Content visible to:** hybris
2. In the Spring **ApplicationContext**, hybris services are of **tenant** scope, not **singleton**.
3. For details refer to [Spring Framework in the hybris Commerce Suite](https://wiki.hybris.com/display/release5/Spring+Framework+in+the+hybris+Commerce+Suite). It leads to problems if you want to inject a service into a component that is of
4. **singleton** scope. The reason is that at the time when a singleton is created, there is no tenant active yet. For Spring Integration, it means you will have trouble using the Service Activator like this:

|  |
| --- |
| <int:service-activator    input-channel="sayHelloProcessChannel"    ref="businessProcessService"    method="startProcess"/> |

1. The Service Activator that is created behind the scenes is in singleton scope, while the **businessProcessService**is in tenant scope. The
2. **businessProcessService** is in tenant scope. The businessProcessService is injected into the Service Activator directly, which can lead to the above problems. To avoid them, use the Spring Expression Language:

|  |
| --- |
| <int:service-activator    input-channel="sayHelloProcessChannel"    expression="@businessProcessService.startProcess(payload)"/> |

1. The expression is evaluated at runtime. At this time a tenant is active, so there is also a **businessProcessService** in scope.

# Platform Basics

* Run administration console, **hAC**, from [***http://localhost:9001***](http://localhost:9001/)*(9002 HTTPS)* by default.
* Start Hybris in hAC ONLY MODE: run **adminserver.bat**. (只是启动hac instead of hybrisserver.bat . As you can see, web contexts are loaded for hAC only)
* Use hAC to view/activate tenants, view/change configuration, set log levels, view extensions, initialize and update system, view license, PK analyzer, CP analyzer (must be switched on with **classloader.monitor.enabled=true** in local.properties);
* **monitor** cluster nodes, cache stats, data sources (with jdbc logging and analyzing), sessions (cancel possible), cron jobs (and abort), jmx beans, jvm memory (gc available), thread dump, run perf tests, run dynaTrace;
* perform **maintenance** by: cleanup (types, search idx, orphan media), encryption, viewing type deployment;
* access **consoles**: BeanShell, Groovy, FlexibleSearch, Imports, Exports, LDAP configuration.
* Access management console, **hMC**, from  [**http://localhost:9001/hmc/hybris**](http://localhost:9001/hmc/hybris)
* Modifications to the types system require **initialization** ((drops and re-)creates type system from scratch into DB from items.xml) or **update** of the system. This functionality can be **locked** from hAC.
* Initialization (**ant initialize**): aborts cronjobs, removes all db tables (in 5.1 not orphans), clears cache, creates media folders, sets licenses; **type system initialized**, create essential/project(optional) data, clears hMC config, localize types.
* **Update**: does not remove tables or remove data, adds new types and modifies old ones, type code cannot be updated.
* To update, run **ant updatesystem** (-Dtenant=X default:master) or from **hAC**. Checking **update running system** rebuilds all type defs. During update orphan types can be created, can be cleaned up from hAC.
* **After init/update all cronjobs are restarted (configurable).**
* Init process is faster (and different) in 5.1. Possible to use old style with property **initialization.legacy.mode=true**
* **Type system initialization is per tenant.**
* Update preserves table name, col name, col type even if changed; does not drop tables or columns; does not delete data; DOES drop and create modified indices; does not change mandatory to optional attribs.
* Type system code change in update treated as new item.
* In 5.1, -Ddryrun=true or via SQL Scripts in hAC generates SQL init and update scripts in hybris temp folder.
* Configuring behavior of hybris (config priority)
  + local.properties = exists in the "config" extension and is the global area for config changes.
    - Changes here take precedent over **any** changes in individual extensions! **Recommended**.
    - extension-specific project.properties
    - Contains extension-specific settings/config
  + Platform project.properties - this should **NEVER** be changed!
  + Application code, programmatic "default" valued params
* Access properties via API (**ConfigurationService**) or Spring ${[property.name](http://property.name/)}.

//fetch the commons configuration instance

**Configuration cfg = configurationService.getConfiguration();**

//there are various convenience methods

String strgValue = cfg.getString("key");

**boolean** boolValue = cfg.getBoolean("key");

**int** intValue = cfg.getInt("key");

BigDecimal bd = cfg.getBigDecimal("key");

//optional default values can be passed as a second parameter

strgValue = cfg.getString("key", "defaultValue");

//listeners can be registered

**final** ConfigurationListener listener = **new** ConfigurationListener()

{

**public** **void** configurationChanged(**final** ConfigurationEvent event)

{

//listener gets invoked before and after the value gets changed

**boolean** isBefore = event.isBeforeUpdate();

Object value = event.getPropertyValue();

String key = event.getPropertyName();

}

};

((HybrisConfiguration)cfg).addConfigurationListener(listener);

#### Adding a Custom Property Placeholder Configurer to Your Extension

* **training-spring.xml**

|  |
| --- |
| <bean id="testBean" class="org.training.TestBean">      <property name="value" value="${custom.property}"/>  </bean>  <bean id="customPropertyPlaceholderConfigurer" class="org.training.CustomPropertyPlaceholderConfigurer">      <property name="properties" ref="hybrisProperties"/>      <property name="order" value="1000"/>  </bean> |

**CustomPropertyPlaceholderConfigurer.java**

**package** org.training;

**import** java.util.Properties;

**import** org.springframework.beans.factory.config.PropertyPlaceholderConfigurer;

**public** **class** CustomPropertyPlaceholderConfigurer **extends** PropertyPlaceholderConfigurer

{

@Override

**protected** String resolvePlaceholder(**final** String placeholder, **final** Properties props, **final** **int** systemPropertiesMode)

{

**if**("custom.property".equals(placeholder)){

**return** "custom.value";

}

**return** **null**;

}}

#### Accessing by Spring

hybris properties are provided in the global **ApplicationContext** of the hybris Commerce Suite. With this, you can inject a specific value by using the usual Spring notation:

|  |
| --- |
| <bean class="Foo">    <property name="bar" value="${key\_of\_property}"/>  </bean> |

The properties are provided by **PropertyPlaceholderConfigurer** with **hybrisPropertiesConfigurer** id, which gets the **hybrisProperties** bean injected. This bean fulfills the **java.util.Properties** class and holds all properties. You can use this bean in other scenarios too, especially if you want to use the hybris properties in your **WebApplicationContext**, having the global **ApplicationContext** as a parent. For this, define:

|  |
| --- |
| <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">    <property name="properties" ref="hybrisProperties"/>  </bean> |

* Use **ant -Duseconfig=test** to load from **localtest.properties**, **ant -Duseconfig=foo** > **localfoo.properties**

Full installation

* Download full Commerce Suite or selected packages. Unzip close to root folder.
* Use bin/platform/setantenv.bat or setenv.bat, followed by ant clean all to build complete.
* Customize installation in **localextensions.xml** and configure them via **local.properties**(optionally set DB params, or use default HSQLDB).
* hybrisserver.bat [debug] starts hybris. Use adminserver.bat to start in HAC only mode.

Directory Structure

* Binary files and source in **/bin** dir (never change config in this dirs). Complete dir and subdirs replaced with new content in a hybris update.
* Config files stored in **/config** dir. If non-existant during first ant call it is created, question for development or production.
* Data files in **/data** dir (media, lucene, hsqldb). Log files in **/log** dir (server, jdbc). Temp files in **/temp** dir. Custom extension placed in **/bin/custom**

### Administrating and Running hybris

Supported Databases, config folder or how to start the server shouldn't be a mystery to you, please make sure you are familiar with the content of these documents:

[Configuring the Behavior of the hybris Commerce Suite](https://wiki.hybris.com/display/release5/Configuring+the+Behavior+of+the+hybris+Commerce+Suite)

[hybris Server](https://wiki.hybris.com/display/release5/hybris+Server)

Starting the hybris Server

Basically, you can start the hybris Server in these ways:

* From the command line
* As a system service

Installing the hybris Server as a Service

Under Microsoft Windows systems:

* Navigate to the ${HYBRIS\_BIN\_DIR} **/platform/tomcat-6/bin** directory.
* Call the  **InstallTomcatService.bat** file.

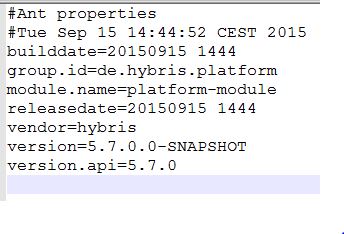
### Removing the hybris Server Service

To remove the hybris Server service:

* Navigate to the ${HYBRIS\_BIN\_DIR} **/platform/tomcat-6/bin** directory.
* Call the  **UninstallTomcatService.bat** file.

Build Framework(3)

* Platform is built together with extensions defined in **localextensions.xml**.
* **localextensions.xml** config
  + **autoload extensions by path (autoload="true"...give path)**
  + lazy load extensions by declaring path - they will load when needed
  + new file can be generated by running: **ant extensionsxml -Dplatform.extensions=**cmscockpit (creates file config/localextensions-generated.xml)
* Performed by "ant" call in /bin/platform (platform scope build) or in /bin/<extension> (extension scope). The entire platform can be built - or only single extensions.
* Commerce suite build number in file **build.number**in /bin/platform.

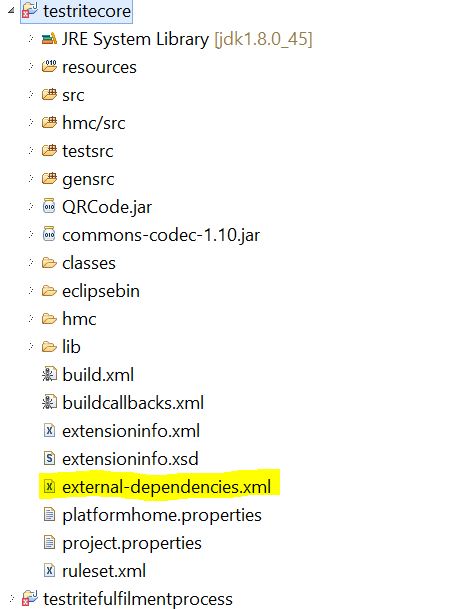


* Build has three phases: preparation, dependency update (5.1), and extension building.
* **Preparation**:
* Checks the directories specified by the **${HYBRIS\_DATA\_DIR}**,**${HYBRIS\_TEMP\_DIR}**, and **${HYBRIS\_LOG\_DIR}** [hybris environment variables](https://wiki.hybris.com/display/release5/hybris+Environment+Variables)and creates the directories if they do not exist.
* Checks the directory specified by the **${HYBRIS\_CONFIG\_DIR}** [hybris environment variable](https://wiki.hybris.com/display/release5/hybris+Environment+Variables) and [prompts for a path if the directory does not exist](https://wiki.hybris.com/display/release5/Configuration+Templates)*.(Create data,log,tempdir if*

*Not exist)*

* Resolves [extension dependencies](https://wiki.hybris.com/display/release5/Extension+Dependencies). By specifying the **<requires-extension="myextension" />** tag in an **extensioninfo.xml** file, you cause that extension to require **myextension**. The consequences are (*checks dependencies in extensioninfo.xml (depdendent built last and its manager called last*);)
  + During the build process, the dependent extension (the extension that depends on **myextension**) is only built after **myextension**has been built. The exact order in which extensions are built is not deterministic, but the build framework abides by extension dependency rules. In other words, you cannot predict exactly when **myextension** is built, but it is always built prior to the dependent extension.
  + During a hybris Commerce Suite initialization or update, **myextension** Manager is called prior to the dependent extension's Manager. That way, the dependent extension can rely on data created by **myextension** Manager to be available (for example, sample products).   
    Again, the exact order in which extensions' managers are referenced is not deterministic, but the hybris Commerce Suite abides by extension dependency rules. In other words, you cannot know when **myextension** Manager is called, but it is always called before the dependent extension's Manager.Refer to [Extension Dependencies](https://wiki.hybris.com/display/release5/Extension+Dependencies) and the documentation on the [extensioninfo.xml](https://wiki.hybris.com/display/release5/extensioninfo.xml) file.
* Prepares the build framework by bootstrapping and copying **build.xml** files into each extension directory.(*copies build.xml to each extension)*
* Generates source files for all extensions (for example, abstract [Jalo Layer](https://wiki.hybris.com/display/release5/Jalo+Layer) classes based on **items.xml** files) *(generates src and models)*
* Generates [Models](https://wiki.hybris.com/display/release5/ServiceLayer+Architecture#ServiceLayerArchitecture-models) for use with the hybris [ServiceLayer](https://wiki.hybris.com/display/release5/ServiceLayer)
* **External Dependency Update During First Build**

To use external library download, update your external-dependencies.xml



To use external library download, update your external-dependencies.xml  (this is where all Maven managed dependencies should be described) and unmanaged-dependencies.txt (this is a list of libraries that are not managed by Maven, like extension specific jar files). Also, change the value of the flag **usemaven** to **true**in the extensioninfo.xml file ( **extension** section)

Because external dependencies are no longer part of platform, the first build will take some more time to complete: during the first build the platform downloads dependencies from the hybris repository.

The ant updateMavenDependencies task deletes all **\*.jar**files from the **lib**folder by de

* **Building Extensions**

A full hybris Commerce Suite build iterates through all available [extensions](https://wiki.hybris.com/display/release5/About+Extensions)and builds them in a **non-deterministic** order that complies with extension dependencies. If this phase is run explicitly by calling **ant** in an extension directory, only that one extension is built.

Before the extension is built, the build framework processes the **before\_build** [callback target](https://wiki.hybris.com/display/release5/Build+Framework#BuildFramework-callback). Then, the build framework performs the following tasks:

* Runs some validations
* Generates source code files if necessary
* Starts compiling the actual extension, which compiles the following modules:
* Extension core module,
* Extension web module,
* Extension hybris Management Console module
* Runs the **after\_build** [callback targe](https://wiki.hybris.com/display/release5/Build+Framework#BuildFramework-callback)

(Therefore, hybris recommends **always running a full build** after a hybris Commerce Suite download so that the build framework components are available to all extensions.)

During an extension build, the hybris Commerce Suite generates Java source files based on the type definitions of the extension in the**items.xml** file. There are different kinds of files that are generated:

* [Models](https://wiki.hybris.com/display/release5/Models) for the [ServiceLayer](https://wiki.hybris.com/display/release5/ServiceLayer)
* Abstract [Jalo Layer](https://wiki.hybris.com/display/release5/Jalo+Layer) classes (carrying a prefix, such as **Generated**). These files are generated anew if you have modified the **items.xml**for the extension. For details, refer to the [Type System documentation](https://wiki.hybris.com/display/release5/Type+System+Documentation).
* Non-abstract [Jalo Layer](https://wiki.hybris.com/display/release5/Jalo+Layer) classes. These are only generated when the file does not yet exists. If a file with the same name exists, the file is not generated anew.

The Java source file is compiled automatically only if the Java source file timestamp is newer than the compiled class file timestamp.

* **Ant Targets**
* Use **ant -p** to see all available build commands.
* The hybris Commerce Suite allows an ant parameter that reads in another properties file after the **local.properties** file(编译替换)
* Advanced: Using Different local.properties Files
* hybris Commerce Suite additionally allows overriding the **local.properties** file. By specifying the **useconfig=$override** parameter in an ant call, the hybris Commerce Suite reads in the file with the name **local${override}.properties**.
* For example, the following ant call uses the **localtest.properties** file:

|  |  |  |  |
| --- | --- | --- | --- |
| C:\hybris\bin\platform>ant -Duseconfig=test | | | |
| **Ant parameter specified** | **File used** |
| **-Duseconfig=myconfig** | **local*myconfig*.properties** |
| **-Duseconfig=test.properties** | **local*test.properties*.properties** |

* **Extending the Build Framework**

To include a custom build logic, edit the **buildcallbacks.xml** file that exists in each extension main directory. There are two ways for the**buildcallbacks.xml** file to affect the platform build process: predefined **macrodefs** and **global ant targets**.

* Customize build using **buildcallbacks.xml**. Use**macrodefs** (eg. catalog\_after\_clean) to perform custom tasks; use **global ant targets** which are then "globally" available (in contrast to extension **build.xml**).
* build hooks
* before/after compile
* Use **ant customize**to replace files in hybris bin dir. Put corresponding files under **/config/customize/** dir.(放在这个目录下面)
* When is "ant all" vs. "ant clean all" necessary?
  + **"ant clean all"**
    - * When adding new types
      * When adding a new extension
  + "**ant all"**
    - * Most other changes (localization, properties file changes, jsp changes, etc)
* When updating any \*-items.xml, you must "update" the system so the changes can be pushed to the DB

Type System

* Types are templates for objects (every object stored in platform is a type instance). Defined in items.xml (or hMC but discouraged).
* Types define
  + **attributes**,
  + database**deployment**,
  + Java**classes** for instances of the type.
* hybris Type = Java Class
* hybris Item = Java Object
* An instance of a **Type** is an **Item** in Hybris as compared to Class vs Object in Java. The **attributes**compare to **fields**.
* Two kinds of types: **system-related** (make up or extend types system) with infrastructure types (ComposedTypes) and data types (collection, map, enum..); **business-related**(order, discount).
* Definitions of Types are stored as instances of **ComposedType**. Can be customized using **metatype** attribute of item XML element.
* The following type (definitions) exist: AtomicType, CollectionType, EnumerationType, MapType, RelationType, ItemType/ComposedType.
* **AtomicType** (Java Integer and String), lacks "code", instead java.lang.String is unique id.
* **CollectionType**, has "code", stores single type, based on Java Collection, List, Set, SortedSet.  
  List of PKs to contained type(1:m), or (m:1) with getter using FlexSearch. Unidirectional.  
  Opposite side of collection is not aware of the CollectionType  
  Data loss (truncation)
* **EnumerationType**, whose values are the instances.
* **MapType**, key/values.
* **RelationType** (n:m), encouraged over CollectionType.  
  **LinkItem** holds **SourceItem** and **TargetItem** attributes. Bi-directional.  
  Stored in separate table.
* Attributes can be: ComposedType; atomic type (Java type, String). Can have: localized name and description; default value.
* New attributes can be added to a type by extending the type, or adding it to an existing type.
* If moving a type to another extension, typecode must not change; java classpath (extension dependencies) must be consistent.
* Performing an update on the type system may result in **orphaned**types that can no longer be reached.
* How to Add a Model: add itemtype in items.xml, build platform, update system!
  + Extend existing types (e.g. Product) extends="Product"
  + New type (implicitly extends from GenericItem)
* [Type System Q&A](evernote://wiki.hybris.com/view/14342068/s124/4c73d6e7-967e-40cf-a777-9220720021b6/4c73d6e7-967e-40cf-a777-9220720021b6/)
* Create new instances of items by:
  + modelService.create() - loads defaults
  + new MyType() - this way does **not** set the defaults...you must manually set them!
* Type System Localization
  + Stored in extension-locales\_XY.properties
  + Can use generator from hMC to get all properties keys generated for selected extension
* Types are lazy loaded - only primitives are loaded from DB, relationships are loaded as-needed
  + "atomic" is the default
  + "none" can be set - and in some cases, might improve performance!

Items.xml

* **Items.xml** defines types using the items.xsd schema. Located in /resources folder and named <extension>-items.xml.
* Has the following structure: atomictypes, collectiontypes, enumtypes, maptypes, relations, itemtypes. Structure MUST be in this **order** and file is parsed top to bottom, so order is important.
* Changes to items.xml are visible immediately in Eclipse, but not in the data model...
* Attributes can be added to types by extending the type (extends="SuperType"), or adding attribute directly to existing type (autocreate="false" generate="false"). Possible to add new attributes directly to hybris types (e.g. Product), but it is **not** recommended (creates direct dependency on core type)! Extend the type instead!
* Models are POJO representations of type system items.
* Prevent items and attributes from being created by model generation process by adding: <model generate="false"/>
* "false" on attribute means no getter/setter for this. "read","write" mods override this.
* Database column type can be changed using <columntype [database="dbX"]> inside <persistence> element, and update system. No col type set will result in default types per DB.
* Items.xml modifiers:
* =true: the type/attribute will be created during initialisation, creates a new database entry for the type at initialization/update process. Setting the **autocreate** modifier to **false** causes a build failure - the first definition of a type has to enable this flag!;

<attribute qualifier=*"lowestInstallmentPrice"* type=*"java.lang.Integer"*>

<modifiers read=*"true"* write=*"true"* optional=*"false"* /> **<defaultvalue>Integer.valueOf(1000)</defaultvalue>**

<persistence type=*"property"* />

</attribute>

<attribute qualifier=*"format"* type=*"java.lang.String"* >

<modifiers read=*"true"* write=*"true"* optional=*"false"*/>

**<defaultvalue>"21"</defaultvalue>**

<persistence type=*"property"* />

</attribute>

Enumerations

***testritecore-spring.xml(***两种方法取值***)***

<bean id=*"colorFacetDisplayNameProvider"* class=*"com.testritegroup.ec.core.search.solrfacetsearch.provider.impl.ColorFacetDisplayNameProvider"*>

**<property name="enumerationService" ref="enumerationService"/>**

<property name=*"i18nService"* ref=*"i18nService"*/>

<property name=*"commonI18NService"* ref=*"commonI18NService"*/> </bean>

* ***Use via***

enumerationService.getEnumerationValue( "enumSampleType" , "sample1" );

* ***Or via***

final HybrisEnumValue colorEnumValue = getEnumerationService().getEnumerationValue(SwatchColorEnum.class, facetValue);

* Localized in **/resources/localization** using [**type.enumSampleType.sample1.name**](http://type.enumsampletype.sample1.name/)=Blah1
* Set default value in the enumtype element or at an attribute in items.xml:
* <itemtype code=*"PackageEntry"* autocreate=*"true"* generate=*"true"*>

<deployment table=*"PackageEntry"* typecode=*"11032"* />

* <attributes>
* <attribute qualifier=*"consignment"* type=*"Consignment"*>
* <modifiers read=*"true"* write=*"true"* search=*"true"* optional=*"true"* />
* <persistence type=*"property"* />
* </attribute>
* <attribute qualifier=*"trackingId"* type=*"java.lang.String"*>
* <modifiers read=*"true"* write=*"true"* search=*"true"* optional=*"true"* />
* <persistence type=*"property"* />
* </attribute>
* <attribute qualifier=*"status"* type=*"TrackingStatus"*>
* <modifiers read=*"true"* write=*"true"* search=*"true"* optional=*"true"* />
* **<defaultvalue>em().getEnumerationValue("TrackingStatus","READY")</defaultvalue>**
* <persistence type=*"property"* />

</attribute>

* Enums can be dynamic, in which case they are no longer Java enums, but can be added to at runtime. Add **dynamic="true"**.

Type Deployment

* Every **instance** of a type is stored in**one row** in DB.
* Type's DB table = "**type deployment**", specified in items.xml. Deployment is **inherited** from super type(s).
* If no deployment specified, the deployment will be to **GenericItem** table. Recommended to specify deployment for subtype of GenericItem, but not "sub-sub-types" for performance reasons (especially during long and complex database transactions, such as synchronization between catalog versions).
* Specify using <**deployment** table="tablename" typecode="typecode\_number"/>. **Type code** must be > 10000, if in use build will fail.
* Deployment for n:m relations must be set, or build fail.
* See **bin/platform/ext/core/resources/core-advanced-deployment.xml**for deployments in different DBs.

Architecture

* Commerce Suite packages: Commerce, Content, Channel, and Orders, depends on Platform.
* Packages consists of modules/extensions. Extensions part of Platform are called core extensions.
* UI Layer: Cockpits, hmc, WS, Stores... ZK or JSP etc.
* ServiceLayer: Models live here (POJOs).
* Type Layer: Types from items.xml.
* Persistence Layer: Store items in DB.
* Database: external. (HSQL, MySql..)
* Commerce Suite runs in single node (single JVM), clustered (multiple JVM), multi-tenant mode (single instance of hybris, multiple sets of data inside one JVM against same DB).

# Basic Architecture

From a business point of view, the hybris Commerce Suite is divided into individual packages, such as Commerce, Content, Channel, and Orders. These packages are bundles of functionality assembled for a certain range of business functionality. All of these packages rely on more basic functionality provided by the hybris Platform. While the hybris Platform can run without any package, no package can run without the hybris Platform.

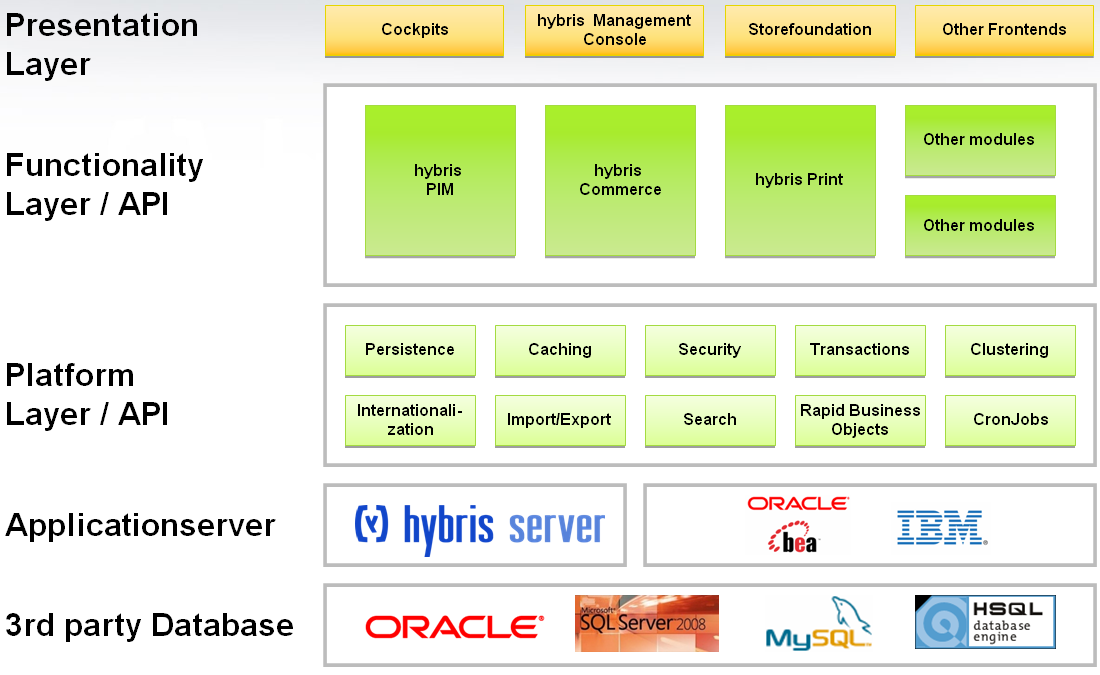
|  |
| --- |
| https://wiki.hybris.com/download/attachments/141792947/cake%20-%20overall.png?version=2&modificationDate=1366989046000&api=v2  Figure: The hybris Commerce Suite diagram |

From a more technical point of view, packages consist of individual modules (also referred to as [extensions](https://wiki.hybris.com/display/release5/About+Extensions)). For example, the hybris Print technically consists of two extensions: **Print** (the technical foundation) and **Print Cockpit** (the graphical user interface).

Extensions are written by hybris or the implementation partner of your project. Extensions written by hybris provide standardized functionality and are supported and maintained by hybris. If you write an extension, you need to maintain them by yourself, but you are free to implement any business functionality you need. A full hybris Commerce Suite installation therefore consists of the hybris Platform plus any hybris packages plus any additional extensions that you have implemented.

Extensions that are part of hybris Platform proper are also referred to as the core extensions. On top of these core extensions, hybris Platform contains several pieces of hybris software, such as the [Build Framework](https://wiki.hybris.com/display/release5/Build+Framework), and third-party software, such as the pre-bundled Apache Tomcat.

hybris Commerce Suite is run in a Java Virtual Machine on a Servlet Container or a J2EE-compliant application server (such as IBM Websphere or Oracle WebLogic) and connects to an external database (MySQL, Oracle DB, Microsoft SQL Server). Internal caching and persistence mechanisms allow the hybris Commerce Suite to run on a Servlet Container. A full-fledged J2EE-compliant application server can be used but is not necessary.

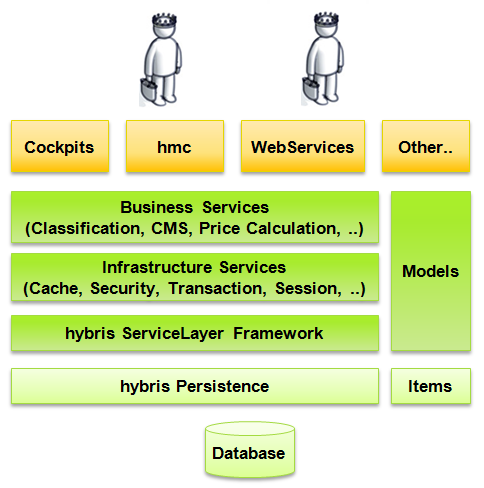
   
Figure: A technical view on the basic architecture of a hybris Commerce Suite installation.

The hybris Platform layer abstracts data from the storage structure on the database using the persistence framework and provides functionality such as [Clustering](https://wiki.hybris.com/display/release5/Cluster+-+Technical+Guide) and the [hybris Platform Cache](https://wiki.hybris.com/display/release5/hybris+Platform+Cache). Relying on the persistence framework, the other functional components of the Platform Layer provide basic business functionality: [Transactions](https://wiki.hybris.com/display/release5/Transactions), [CronJobs](https://wiki.hybris.com/display/release5/cronjob+-+Technical+Guide), [Personalization](https://wiki.hybris.com/display/release5/Restrictions), [Internationalization](https://wiki.hybris.com/display/release5/Internationalization+and+Localization+Overview), and more.

The packages on the Functional Layer (hybris Commerce, hybris PIM, hybris Print) use the hybris Platform to implement the functions they deliver. Actually, hybris Platform is part of any hybris Package.

# Layer Architecture

The hybris Commerce Suite contains several layers, each of which has a different function and data abstraction level.

   
Figure: Overview of the architectural layers of the hybris Commerce Suite.

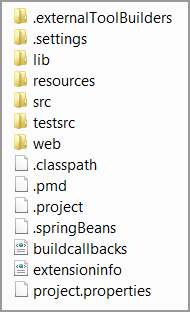
# Modes of Operation

You can run the hybris Commerce Suite in three different modes of operation:

| **Operation Mode** | **Schema Graphics** | **Description** |
| --- | --- | --- |
| **Operation Mode** | **Schema Graphics** | **Description** |
| Single Node | https://wiki.hybris.com/download/attachments/141792947/cache-single_node.png?version=1&modificationDate=1235575348000&api=v2 | The most basic mode of operation. A single machine running one instance of a hybris Commerce Suite installation. Does not have several nodes (unlike in Cluster Mode) and only has one set of data (unlike Multi-Tenant mode). |
| Cluster Mode | https://wiki.hybris.com/download/attachments/146317082/cache-cluster.png?version=1&modificationDate=1355136152000&api=v2 | [The hybris Cluster](https://wiki.hybris.com/display/release5/Cluster+-+Technical+Guide) consists of several individual nodes. These nodes access a common database and communicate among each other via the TCP or UDP protocol. Summing up, this is a multi-node, cross-linked version of Single Node. |
| Multi-Tenant Mode | https://wiki.hybris.com/download/attachments/141792905/cache-multi_tenant.png?version=1&modificationDate=1219078950000&api=v2 | A hybris Commerce Suite in [Multi-Tenant Mode](https://wiki.hybris.com/display/release5/Multi-Tenant+Systems) allows using several individual, distinguished sets of data separated by database table prefixes. Multi-Tenant Mode can be used for Single Node and for Cluster Mod |

Extensions(7)

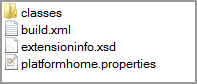
* **Extensions** are encapsulated modules with **business logic**, **type definitions**, a **web application**, or a **hMC configuration**.
* Extensions can be created using **extension templates**, such as **yempty**, or **yacceleratorstorefront**.(ant clean 或是exstion创建以后有以下结构)



Directory structure:

* + /.externalToolBuilders: Eclipse auto build on items.xml change.
  + /.settings: Eclipse settings.
  + /lib: external libraries.
  + /resources: Contains extension-items.xml and the /localization dir for types (and other resources).
  + /src: Source code for the extension's core module.
  + /testsrc: JUnit tests.
  + /web: Web app resources (JSP, JS..). Resources cannot be used by another extension.
  + .classpath (eclipse); .pmd (hybris); .project (eclipse); .ruleset (pmd); .springBeans (spring IDE); buildcallbacks.xml (build system);
  + project.properties (extension specific props)
  + **extensioninfo.xml:** Configures the **core**, **web** and **hmc** modules of the extension and **dependencies** to other extensions.
  + /classes: generated, contains .class files for src, testsrc, and gensrc.
  + **build.xml** (generated by build); extensioninfo.xsd (used for validating extensioninfo.xml); platformhome.properties (relative path to /platform).

**During an extension build, some additional directories and files are created(**编译后产生的几个文件**)**



|  |  |  |
| --- | --- | --- |
| **/classes** directory | Contains all **.class** files for **src** , **testsrc** and **gensrc** folders of the extension. |  |
| **build.xml** file | This build file calls the platform's build file for actual action and provides the necessary parameters to it. This way **ant**targets are consistent throughout the entire installation. |  |
| **extensioninfo.xsd** | Copied from *${HYBRIS\_BIN\_DIR}* **/platform/resources/schemas** . Allows validating the **extensioninfo.xml** file. |  |
| **platformhome.properties** | Automatically generated file. It contains only the relative path to*${HYBRIS\_BIN\_DIR}* **/platform** . Do not modify it. |  |

* **List of all extensions in hAC**: platform->extensions. Shows version, modules and webroot (if avail).
* **Core** extension module: Declarations and localizations of types, Jalo item classes. in **<extension>-items.xml**
* **hMC** extension module: hMC configuration and localization, custom hMC elements, such as editors, tabs. In /hmc dir. Not created using extgen in v5+!
* Web extension module: Web application. Default under http://localhost:9001/<extensionName>
* Extension version: in /resources/<extension>.build.number.
* To add an extension to hybris**, modify localextensions.xml** in the global /config dir, and rebuild.
* **testsrc** **is not part of an extension module. Does not need to be in extensioninfo.xml. Can be run** with **ant yunit**.
* External libraries: in /lib visible to other extensions; .../WEB-INF/lib**: visible to web app only.**
* Create extension using ant target **extgen**. This is based on templates.
* Custom templates in extensioninfo.xml   
   ***<meta key="extgen-template-extension" value="true"/>***
* **Adding a Custom Template**
* To add your own template extension to the selection of extgen templates please add the following line to the **extensioninfo.xml** of your template extension.

|  |
| --- |
| *<meta key="extgen-template-extension" value="true" />* |
|  |

* Steps to **create extension**:
* *Optionally: Modify default values for the extension generation process.*

**ant extgen**

**Extgen** prompts you to specify (Extgen asks for extension name, Java package, template to use. Modifies template, and then)

The ***extension's name.***

The ***extension's Java package***.

The ***extension template to use***.   
For a list of available extension templates compatible with **extgen**

* *Create a new extension.*
* *Reference the new extension in thelocalextensions.xml file.*
* *Rebuild the hybris Commerce Suite.*
* *Update the hybris Commerce Suite.*
* 3) Reference the extension in
* copies to new extension location. Default /bin/custom/<extension>.
* **localextensions.xml**

|  |
| --- |
| <path autoload="true" dir="${HYBRIS\_BIN\_DIR}/ext-platform"/>  <path autoload="true" dir="${HYBRIS\_BIN\_DIR}/ext-cockpit"/>  <path dir="${HYBRIS\_BIN\_DIR}"/ |

* Available templates:

Technology Templates package is a set of hybris plugins that enable you to create custom-specific solutions. The following Technology Templates are available:

* [ybackoffice](https://wiki.hybris.com/display/release5/Structure+of+the+Custom+Backoffice+Extension): Template to generate your backoffice extension to start developing your widgets.
* [ycockpit template](https://wiki.hybris.com/display/release5/ycockpit+Template+-+Technical+Guide): An example of cockpit plugin
* [ycommercewebservices](https://wiki.hybris.com/display/release5/ycommercewebservices+Extension): *It provides a working example how a REST API can be exposed.*
* **yacceleratorfulfilment process**extension
* **yaddon**extension: It is a base template for writing AddOns.
* **yempty**extension
* yempty, ycockpit, yaddon, yinstoreinitialdata, ybackoffice, yaccelerator\*,yb2baccelerator\*, ytelcoaccelerator\*, ycommercewebservices.

MCC

* Multichannel Commerce Cockpit, contains links to all other web extensions.
* Links can be **static**: created in hMC or via impex (StaticLink type). Can define "**area**" (platform, channel, commerce, content), sub links and parent links, and have access rights applied.
* Links can be **dynamic**: Extends static links with beanshell scripts for dynamically defining links. hMc or Impex: DynamicLink type.
* **Dividers** (Divider type) can be used to add empty space with optional horizontal bar, and are defined for a specific area.

hMC

* hMC config can be stored on the file system or the database (**hmc.structure.db** in local.properties)
* If db:
  + hMC's StructureLoader fetches layout def from db, otherwise uses last known build from file system
* Can upload the file manually (System – > hMC Configuration)
* hMC localizations are stored in locales\_XY.properties files in the extension hmc/resources/${package}/hmc directory
* Accessed at <http://localhost:9001/hmc/hybris>. Log out via session timeout, or logout button.
* **Toolbar** area: New business object; back/fwd buttons; close session; edit user; quick search (not default in v5+); lang selection.
* To use quick search, add **lucenesearch** and **lucenesearchhmc**extensions during build.
* Default Tree: System; Catalog; Multimedia; User; Order; Price Settings; I18N; Marketing; Cockpit.
* Left click: show in Organizer (right window); Right click: show context creation menu to create new object.
* Organizer areas: Search, Result, Edit. Organizer areas' layout depend on the type of business object.
* Different **search conditions** can be applied to type attributes, and are AND:ed. (For products you can also search for Categories (classifying).
* Result area types: List, Tree, Edit (as list but items editable directly). Depends on business object.
* **Actions** can be performed on multi-selection in result area via toolbar or right-click. Advanced actions via right-click -> Action.
* A types organizer can be reached from Editor area upper right-hand corner.
* **Editor** Area business objects' attributes are grouped into sections, which are grouped onto tabs. Visibility depend on access rights of user.
* Attributes are defined by **Attribute Editors** and have different appearence (String/StrLocalized, Boolean/Advanced, List).
* Unsaved changes are lost when session ends, or a new search is run for same type.
* A **Wizard** is an editor area that acts as a guide to creating business object in predetermined steps. (Example start synchronization wizard.)
* **Localized attributes** can be expanded to see values for all langs. Inactive locales are visible in hMC (but may not be in effect).
* What langs are displayed can be set via **User Profile**. The user profile can set read/write language right. For localized attribute editors only!
* Edit user profile via the **Language Settings** button in the top right end of the hMC Toolbar.
* Recent Items Area in the bottom left corner. Not persisted, and cleared at session end.
* Browser back and forward buttons do not operate correctly in hMC due to the use of AJAX.
* hMC customization via an extensions**hmc.xml**. Must conform to hmc.xsd or build fails.
* hmc.xml contains three major parts: <**action**> "toolbar actions"; <**explorertree**>; <**type**> "type Organizer".  
  Root element is <**configuration**>.
* Action classes should implement **HMCAction** class.
* <**defaultattributemappings**> can be used to override default attribute editor and display chips.
* <**defaultmappings**> can be used to override core hMC classes such as ToolbarChip or OrganizerListChip.

ImpEx(3)

Use ‘append’ mode to avoid overriding existing references

INSERT\_UPDATE Employee;uid[unique=true];groups(uid)[mode=append]

;FrankColumbo;approvers,dummygroup,reviewers

Use ‘translators’ for custom interpretation of imported values

INSERT\_UPDATE Employee;uid[unique=true];@password[translator=PasswordTranslator]

;FrankColumbo;aVeryStrongPassword;

INSERT\_UPDATE Media;code[unique=true];@media[translator=MediaDataTranslator]

;media01;/path/to/my/picture.jpg;

Batch update

UPDATE Product [batchmode=true]; itemType(code)[unique=true];status

;Product;approved

ImpEx Syntax and Examples |

In live operation:

. to import customer data into a production system (可以直接到如数据到生产系统)

. to synchronize data with other systems, such as an ERP or LDAP (同步ERP和LDAP的数据)

. to create backups (很好的备份数据)

. to update data at runtime (运行时更新数据)

. can be run from CronJobs (通过job更新数据)

* Use for initial data, testing, migrations, synchronization, backup...create, update, export and remove items
* Import from hMC (System-Tools-Import), hMC cronjob (System-Cronjobs, right click "New **ImpExImportCronJob**"..), ImpEx page in hAC (development only), or via API (Jalo and ServiceLayer).
* Since 4.5 release ImpEx import is based on the ServiceLayer, thus the ServiceLayer infrastructure like interceptors and validators is triggered.  
  Unless ImpEx is run in legacy mode.
* **Importer** class or **ImpExManager** class. Both import via Cronjobs.
* Two import validation modes, strict and relaxed.
* Three export validation modes, strict, relaxed, only (cannot be reintegrated).
* Possible to create ImpEx abbreviations in local.properties.
* Export from hMC, hMC cronjob, ImpEx page in hAC (development only), or via API.
* Exporter class or ImpExManager class. Both import via Cronjobs.
* ScriptGenerator in hMC (System-Tools-Script Generator) can generate export script for all types.
* Export script structure is: set target file (BShell), export header, data to export (BShell).
* Use **cellDecorator** to change parse to translation behavior (parse, decorate, then translate). ;myAttr[cellDecorator=myDecoratorClass]  
  Must implement **CSVCellDecorator** interface
* Use **translator** to change translation behavior (after decoration): ;myAttr[translator=myTranslatorClass]
* BeanShell, accessed by #% (enclosed with double quotes). Disabled by default!  
  "#% import de.hybris.platform.core.Registry;"  
  "#% impex.enableCodeExecution(true);"
* Use **includeExternalData** from BeanShell to include other import scripts for structure.  
  "#% impex.includeExternalData(ImpExManager.class.getResourceAsStream(""/importdata.csv""), ""utf-8"", 0, 0 );"
* Include data via beanshell by: include resource stream, reference platform media, from database.
* Available BeanShell vars include "impex" (**ImpExReader/**ImpExImportReader) and "line" (Map with line data).
* Control structures: beforeEach,beforeEach:end, afterEach,afterEach:end, if, endif (latter two can be nested).
* Use [impex.info](http://impex.info/)(), impex.warn() to log statements.
* Use $START\_USERRIGHTS and $END\_USERRIGHTS to modify access rights and create users.
* There are built-in standard value translators, and also special value translators (MediaDataTranslator, UserPasswordTranslator...).
* ImpEx syntax include lines: header, macros, values, comments, beanshells, userrights.
* **INSERT** inserts a new item, does not check for existing items unless unique modifier(s) is used.
* **UPDATE**updates existing item, dumped if not found. Default values for unspecified values is NULL.
* **INSERT\_UPDATE**combination of insert and update.
* **REMOVE** removes an item (or logs warning if not exists).
* Each line can specify subtype of the header type.
* References to attributes (e.g. ComposedTypes) can be done with PK, or by specifying target type attribute in the header: unit(code).
* Use colon to recurse: catalogVersion(catalog(id), version) ->**clothescatalog:Staged**
* Localizing is done with **[lang=en]** attr modifier. Localization for languages not specified is NOT overridden.
* Special attributes can be translated using the "@" symbol together with translator class.
* Header-related modifiers exist, where e.g. you can specify batchmode=true, to make changes to multiple items with one impex line.

UPDATE Trigger[batchmode=true];itemtype(code)[unique=true];maxAcceptableDelay ;Trigger;3600

* Attribute-related modifiers exist, e.g. unique, to specify which items to update (or if missing, insert).
* Comment lines start with a #.
* Macro lines: $catalog=catalog(id)
* ImpExing can be multi-threaded. E.g. **impex.import.workers=8** in **local.properties**.
* Cannot use transactions
* ImpEx media in zip file can be imported.
* If an ImpEx fails, running it as a Cronjob can help troubleshooting by setting log levels.
* BeanShell scripts are not run during dumped runs.
* By default the first line in included scripts is ignored (configurable).
* During initialization and update, import data is found in extname/resources/impex folder. essentialdata\*.impex, and projectdata\*.impex.
* Can be configured e.g. **by extname.essential-data-impex-pattern=**... property.
* Failed lines are retried (so order doesn't matter)...validation can be disabled
* Use "append" mode to avoid overriding existing references

Cronjobs (2)

* Summarizing, a **Job** defines what should be done, a **Trigger** says when, and a **CronJob**specifies the setup used by **Job**.
* Each **Job** has a **JobPerformable** which defines the actual logic.(实现JobPerformable 接口)
* Can start via ImpEx, hMC (scheduled / on demand), Ant or using the API
* Abort cronjob
* **CronJob** type has an attribute named **sessionUser** that holds a single user, by default this is same user that created job.
* **CronJob** session attributes: user, language, currency
* A cronjob has a **status** **(FINISHED etc**) and **result** (**SUCCESS etc**).
* Several cronjobs can be run in sequence with **CompositeCronJob**.
* In a cluster, a cronjob is always **run on ONE single node** (configurable which one).
* Restrict cronjobs with **JobSearchRestrictionModel**.
* A **CleanupCronJobStrategy** exists to clean up old, finished cronjobs.
* Cronjob can be extended by extending **AbstractJobPerformable** or implementing **JobPerformable**
* Disable cron jobs with

cronjob.timertask.loadonstartup=false in local.properties

How do you create a cron job?

MergeHCTDeliveryNotesJob **extends** **AbstractJobPerformable**<CronJobModel>

@Override

**public** PerformResult perform(**final** CronJobModel arg0)

{

**return** **new** PerformResult(CronJobResult.***SUCCESS***, CronJobStatus.***FINISHED***);

}

cronJobService.performCronJob(myCronJobModel, true);

cronJobService.requestAbortCronJob(myCronJobModel);

How can you trigger it?

* **ant runcronjob**
* **cronJobService.performCronJob(myCronJobModel, true);**

#Merge-Delivery--files CronJob

INSERT\_UPDATE CronJob;code[unique=true];job(code);singleExecutable;sessionLanguage(isocode)

;**mergeDeliveryHCTJob;mergeDeliveryHCTCronJob(Spring-ID);false;de**

INSERT\_UPDATE Trigger;cronjob(code)[unique=true];cronExpression

**;mergeDeliveryHCTJob; 0 0 16 \* \* ?**

cronjob.trigger.interval=30

Flexible Search (5)

* FS is an abstraction of SQL queries, and operates at service layer level.
* FS queries are pre-parsed into SQL which is then run (two phases).
* queries are cached!
* Example with localization: SELECT {name[de]}, {name[en]} FROM {Product}
* **FS query results are user dependent (which user runs the query), by default anonymous.**(缺省是匿名的)
* FS queries contain **SELECT, FROM, WHERE**and**ORDER BY**(optional, 'asc' default).
* Basic syntax: "SELECT <selects> FROM <types> (WHERE <conditions>) ?(ORDER BY <order>)?
* Mandatory: "SELECT <selects> FROM <types>
* To avoid searching in subtypes, use **exclamation mark**:   
  SELECT {code},{pk} FROM {Product!}
* Parameters to queries: LIKE ?product
* FS can be run using FS service:   
  SearchResult<ProductModel> searchResult = flexibleSearchService.search(query);
* Call List<CustomType> res = searchResult.getResult(). This is **lazy** loaded, beware!
* To pass **parameters**, create a Map and pass it like this:   
  flexibleSearchService.search(query, params);
* If more than one db colum is retrieved, **several entries per row are returned and the returned item ~~cannot~~ be cast to the correct type. .**(不同entry 返回不同的item有可能隐射不同)
* FS can be queried from hMC via **SavedQuery**(improvement to General Search) and **ViewType** (e.g. database view style).
* **Pagination** via **FlexibleSearchQuery** customization object with (每次分页都会产生不同的 DB)
  + **setNeedTotal=true** passed to FS service,
  + **query.setCount,**
  + **query.setStart**
* But even with pagination, each retrieved "page"

causes a separate DB query, watch out for performance! **Search results may differ over time!  
Use ORDER BY (or results are random order)!**

* FS queries can be executed from hAC console.
* Restrictions can be added and ends up as additions to WHERE clause:  
  WHERE ... AND {p:description} NOT NULL
* In contrast to FlexibleSearch, a query for the **GenericSearchService** is constructed by combining instances of **GenericField** and **GenericCondition** to form **GenericQuery**. **GenericQuery** is a Java-based object describing the search criteria, an example of which is shown below. Users of Hibernate ORM can see a close parallel between this approach and Hibernate's Criteria Queries.

Beans and Enum Generation

* Beans and enums can be generated by XML configuration, **extname-beans.xml** following beans.xsd, residing in **/resources** folder.
* Bean definitions in multiple extensions are merged into one type, that is also **serializable** and has **no-args** constructor, with **getters/setters.** Thus you don't have to subclass existing type, but can simply extend.
* Generated classes are located in {HYBRIS\_BIN\_DIR}/platform/ext/core/gensrc directory.
* It is possible to specify a Velocity template to control what ends up in source code of the generated type.
* Used to transport objects to the front end layer (DTOs)...contain a subset of attributes

Dynamic Attributes(2)

* DA adds logic to types without modifying models. **They are not persisted in DB and are specified** in items.xml with **persistence type="dynamic".**
* An **attribute handler** bean id is generated for the DA, default with name ModelName\_attributeNameAttributeHandler. One must then implement the class that implements **DynamicAttributeHandler**andadd it to Spring ctx**.**
* Implement **DynamicLocalizedAttributeHandler**if the attribute is localized.

***testritecore-items.xml***

* [Dynamic Attributes](https://wiki.hybris.com/display/release5/Dynamic+Attributes)
* [How To Create Dynamic Attributes - Tutorial](https://wiki.hybris.com/display/release5/How+To+Create+Dynamic+Attributes+-+Tutorial)
* <!-- Dynamic Attribute -->

<attribute qualifier=*"displayName"* type=*"localized:java.lang.String"*>

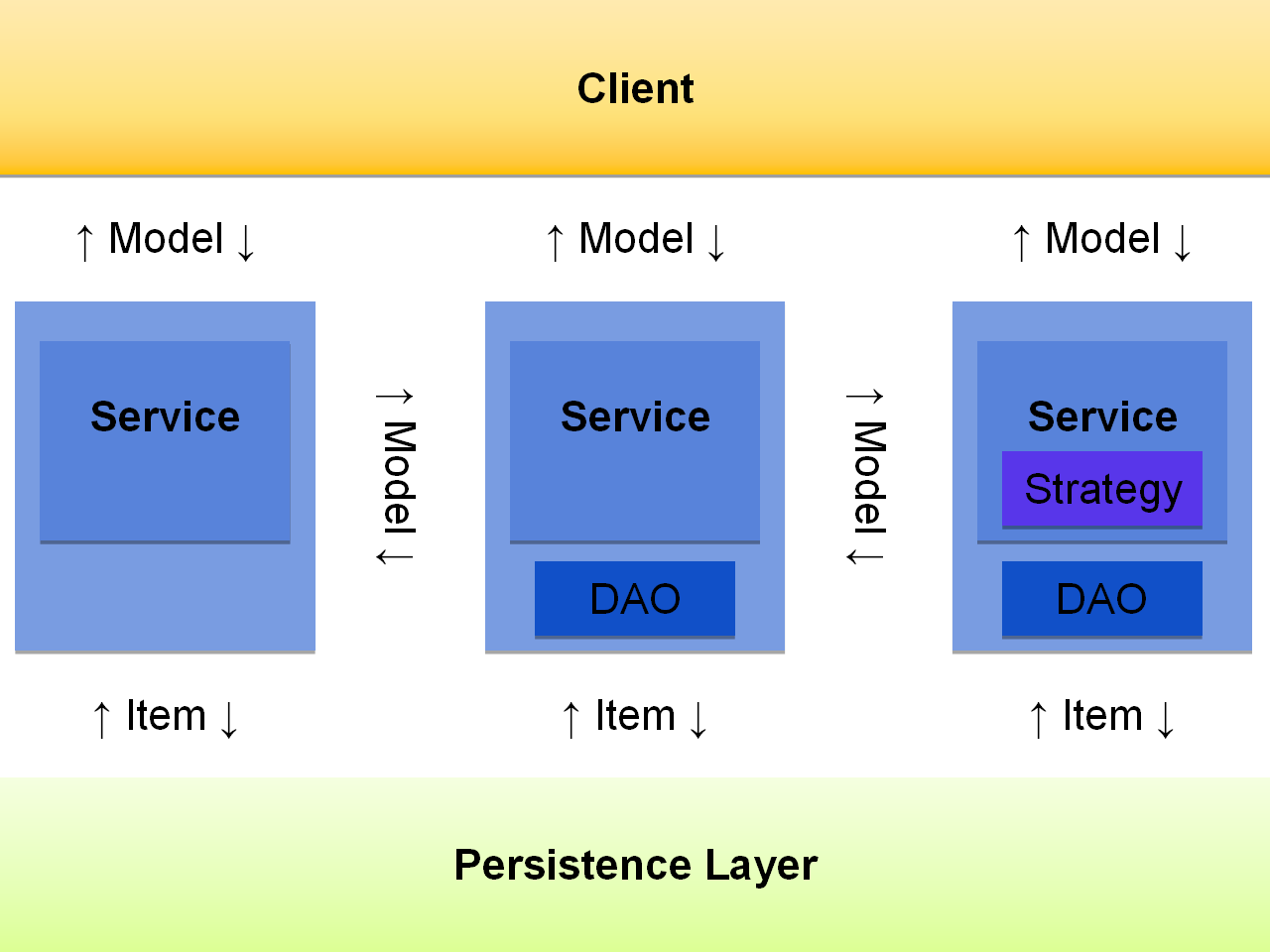
* <description>Product Display Name</description>
* <modifiers read=*"true"* write=*"false"* />
* <persistence type=*"dynamic"* attributeHandler=*"productDisplayNameAttributeHandler"* />
* **public** **class** ProductDisplayNameAttributeHandler **implements** DynamicLocalizedAttributeHandler<String, ProductModel>

Service Layer(4)

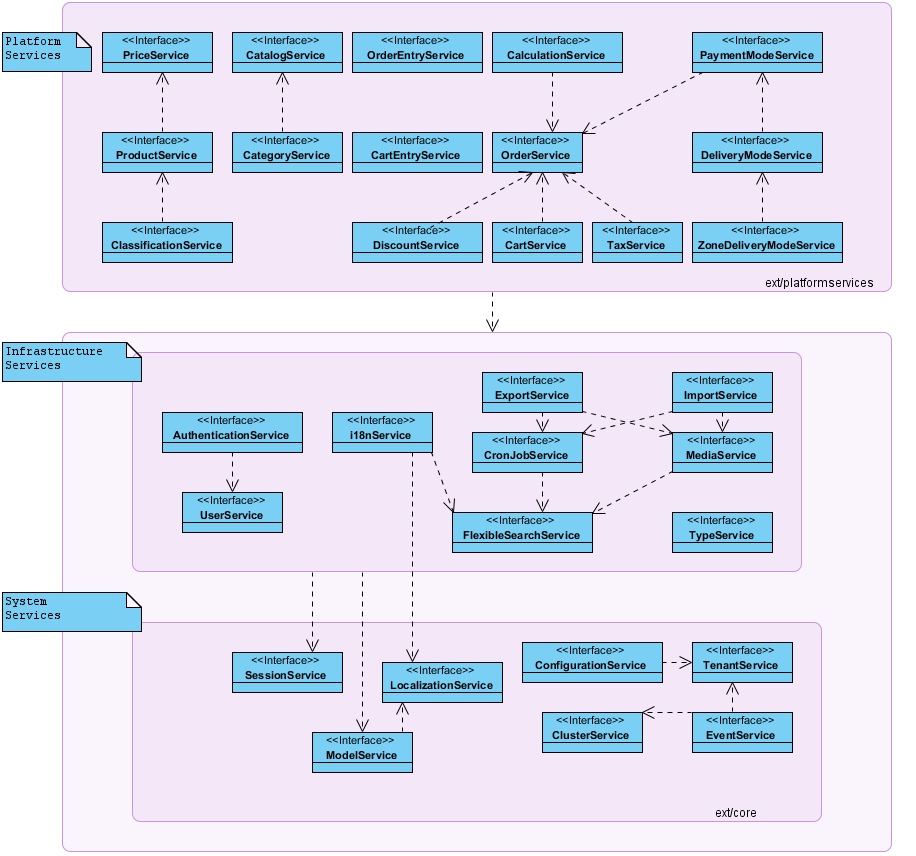
The hybris ServiceLayer is an **API** to develop with and for the hybris Commerce Suite.

The main characteristics of the ServiceLayer are:

* Is based on a service-oriented architecture.
* Provides a clean **separation** of business logic and persistence logic.
* Provides a number of **services**, each with its well-defined responsibilities.
* Provides a **framework** to develop your own services and to extend existing ones.
* Is heavily based on the **Spring** Framework.
* Is based on **common patterns**, such as interface-oriented design and dependency injection.
* Is the layer where partners should implement their business logic.
* Provides hooks into **model life-cycle events** for performing custom logic.
* Provides hooks into system event life-cycle events such as **init and update** process.
* Provides a framework for publishing and receiving **events**.



The Objectives of the Service Layer are to be:

* ***Consistent***
* ***Easily approachable***
* ***Comprehensive***
* ***Adaptable***
* ***Extensible***
* ***Flexible***
* The Service Layer is Spring-based,**SOA-oriented** framework, making a clean separation of
* SL interconnects the persistence layer with the client. It sends data between client in "models" and to and from peristence layer in hybris "items".
* Typically a service consists of a **Service** (facade towards Strategies), a **Strategy** (Delegated business logic), and a **DAO** (data access encapsulation with FlexibleSearch or SQL) working on a **Model**.
* A service can also contain **DTOs**, **Facades**, **Converters**.
* There are **business/platform** services (cart etc), **infrastructure** services (ImpEx etc), and **system** services (session etc).
* List of services: [Key Services Overview](https://wiki.hybris.com/display/release5/Key+Services+Overview)  
  System / Core Services (**ModelService**, **ClusterService**, **ConfigurationService**, etc)  
  Infrastructure / Business Services (**FlexibleSearchService**, **AuthenticationService**, **UserService**, **CronJobService**)
* 
* Use @SystemSetup(extension = MyExtension.EXTENSIONNAME, process = Process.UPDATE, type = Type.ESSENTIAL) on types to **hook into** init and update system.
* **Event System**
  + Provided by the ServiceLayer to send & receive events within hybris - based on the Spring event system
  + Can be published locally or across cluster nodes and can be transaction-aware
  + Event processing is synchronous on same node and asynchronous on other nodes.  
    Make async on same node with **ClusterAware**events (but...tradeoff is network chatter).
* Use the **EventService** (**registerEventListener(),** **publishEvent()**) to use (Spring) events in the service layer.  
  Extend **AbstractEventListener's onEvent()**method and add as Spring bean for auto-inclusion in event system.
* **TransactionAwareEvents** can be useful at the end of a transaction to publish whether a model changed

Interceptors

* Interceptors modify life cycle of **models**.
  + **InitDefaultsInterceptor**, **LoadInterceptor**, **PrepareInterceptor**, **ValidateInterceptor**, **RemoveInterceptor**.
* InitDefaultsInterceptor called when a model is filled with default values (modelService.create()). Set further default values.
* LoadInterceptor called when a model is loaded from DB (modelService.get()). Change values after loading.
* PrepareInterceptor called before model is saved to DB (before Validate Interceptor). Add or modify values before persist.
* ValidateInterceptor called during model validation before model is saved to DB. Validate values and possibly raise **InterceptorException**.
* RemoveInterceptor called before a model is removed. Remove models not related, or prevent removal raising InterceptorException.
* Register custom interceptor as Spring bean in <myextension>-spring.xml. Also add Spring InterceptorMapping bean.  
  Specify interceptor, typeCode, order (of running), replacedInterceptors.

Workflow

* Emulate batch card-like processes, e.g. model interaction flows between people in hybris.
* Basic Procedure
  + Creating a **WorkflowTemplate**
  + Creating a **Workflow** from a **WorkflowTemplate**
  + Processing a **Workflow**
* A **WorkflowTemplate** instance is a template for a real workflow instance, and contains a flow of actions to be executed.
* A **WorkflowActionTemplate**is used for describing a specific task of a workflow. Each action is owned by a **principal**, the user responsible for deciding on the action.
* **WorkflowTemplate** (defines the sequence of events) holds a **WorkflowActionTemplate** (definition of a step in a workflow), which holds an **Option** (possible results of a WAT)
* An **activation script** is used to trigger a workflow, when an item is created/saved/removed.
* Workflows are essentially cronjobs, but they proceed according to **Decisions**on the **Actions** on the flow.
* Admin workflows from hMC **System - Workflow Administration**.

Process Engine

* Enables users to define **business** processes through **XML** process definitions and runs these processes in an **asynchronous** way.
* Processes tasks async, and are distributed across the cluster to ensure reliable task processing
* Somewhat similar to Workflows, but more heavy weight and less for people oriented business processes.
* Process definition file contains **nodes** and **transitions**
* Main elements of a process XML (defined by **processdefinition.xsd**):  
  <process xmlns="<http://www.hybris.de/xsd/processdefinition>" name="Example" start="Action1">...  
  <action id="Action1"bean="Action1">  <transition name="OK"to="Action2"/>…  
  <wait id="waitForWarehouseSubprocessEnd" then="isProcessCompleted"> <event>ConsignmentSubprocessEnd</event>…  
  <end id="success"state="SUCCEEDED">Everything was fine</end>  
  <split id="split">  <targetNode name="rnd"/> <targetNode name="sayC"/>…  
  <notify id="notifyadmingroup"then="split">  <userGroup name="admingroup"message="Perform action"/>...
* Action1.. beans should be defined in Spring cfg.
* Actions are carried out in a certain order and only carried out if all predefined conditions are met
* End actions for events: wait, notify users/groups, fire actions, determine further actions
* For each action, you need to define a class / Spring bean that extends one of two abstract classes:
  + **AbstractProceduralAction**: Simply returns OK whatever happens. It is useful to split a Process into smaller pieces.  
    Only an implementation of the **execute** method is necessary.
  + **AbstractSimpleDecisionAction**: It returns one of OK or NOK values. It is useful to make a simple decision.  
    Transitions are defined and only the **execute** method must be implemented.
* It's necessary to define the process via an instance of **ProcessDefinitionResource** pointing to the process XML, in Spring cfg.
* Create a process by calling:  
  businessProcessService.createProcess(processName)
* Start a process by calling:  
  businessProcessService.startProcess(businessProcessModel)
* Retry functionality is available via  **RetryLaterException**or returning RETRY\_RETURN\_CODE, making the engine fire the action once again.

Tasks

* Lightweight alternative to cronjobs.
* Tasks are scheduled either at a specific time, or triggered by an event.
* Items **Task** and **TaskCondition** are persisted in the database.   
  **TaskService** manages tasks:   
  taskService.triggerEvent("MyEventArrived" );  
  taskService.scheduleTask(task);
* Tasks are "cluster aware", and are processed by the first available node.
* Make implementation of **TaskRunner**, and register the bean in the core/tenant app ctx.
* Expiration time (timeout) and custom context can also be set on tasks.
* Retry functionality is available via  **RetryLaterException**, which causes the task engine to process action at later time.

Catalog

* A **Catalog** holds **CatalogVersion**(s) that holds **Catgory**(s) that hold categories, products or classifications.
* Catalog extension adds its own access rights for hMC. Users need both type rights AND catalog access rights to view/edit catalog, catalogversion etc.
* In web applictions (for customers) the **visibility** of products depend on five things:
  + #1 Catalog URL Pattern. /skateboards will only see products in the skateboards catalog.
  + #2 Catalog version active. Only one active version per catalog.
  + #3 Category is visible. Normally set customerusergroup on root category, then visibility is inherited. Visibility can only be allowed and not denied.
  + #4 Date Range. Visible if the date is in the product Validity Period.
  + #5 Product Approved. Multiple products can be approved via Catalog Version->Content->Show all Products of this Catalog Version and the Action Approve.
* Restrictions must be set in web app via filters. Add one of below filters to PlatformFilterChain bean in Spring:
  + **SimpleCatalogVersionActivationFilter** - Sets configured catalogs on the session.
  + **DynamicCatalogVersionActivationFilter** - Can set catalogs at runtime.
* During system init or update, the Catalog manager makes sure there is a default catalog with staged/online.

Classification

* Classification based attributes are called **category features**, sometimes also referred to as **classification attributes**.
* Classification systems are separate from product catalogs. Each system has classification system versions (with one active) structured into classifying categories.
* Category features are passed down the classification tree, "inherited".
* A feature is defined by category, feature descriptor ("weight" localized), feature descriptor type ("Number"). Values are set on Products, but can also be specified with Feature Values in class system.
* A feature can also have a Classification Unit (defined in class system).
* Several classification systems may be used simultaneously. E.g. a product can have category features stemming from different systems.

Cockpit Extension

* Framework for backoffice frontends in Hybris. Based on ZK.
* The extension can be customized in three fashions: easy (configuration only), medium (using existing cockpit as template), expert (use framework not based on existing cockpit).
* To create a new cockpit, use **ant extgen** with **ycockpit** template. Edit configuration in /resources/<extension>/<extension>-web-spring.xml.
* <extensionname>-web-spring.xml holds:
  + A definition of the UICockpitSession bean that holds some generic information about whole cockpit configuration, and also holds references to available perspectives
  + A definition of your one and only perspective <extensionname>Perspective that holds references to navigation areas that can be displayed here
* Within a cockpit the interface between components and items (e.g. Products) is the **TypeService**, which deals with **ObjectType** (Item Type), **TypedObject** (item instance), **PropertyDescriptor** (attr on Type).
* Cockpit framwork allows jump-in URLs which perform some action based on URL parameters. Address default: /index.zul?events=<event>
* Jump-in URLs can change perspective; activate item; perform search.  
  Custom event handlers extend **AbstractRequestEventHandler**, and are defined in Spring bean **DefaultRequestHandler**.

Cockpit NG

* Cockpit NG framework let's users create **widgets** that can be **orchestrated** into customized **backoffice** applications.
* Delivered with the **backoffice**extension with extension possibilities in **ybackoffice** ext template.
* **localhost:9001/backoffice**
* Each widget is a stand-alone, deployable component with a clearly-defined interface and a specific purpose.  
  Widgets are put together to create an app in the Application Orchestrator (AO).
* AO is opened with **F4** from backoffice app.
* Backoffice extension also contains the Main Backoffice Application, which should be used by everyone and displays relevant aspects by user's business role.
* Widgets are organized in a tree structure. A widget tree represents an application. You can take a widget sub-tree (a widget group) and reuse it in other application, just like a single widget.
* A widget outputs a message or data, and then other widgets receive and act on the information if configured to do so. There are no direct dependencies.
* Widgets can communicate with each other via inputs/outputs called sockets...Sockets have IDs and type defs to help w/checking when connecting two widgets
* **WidgetConnections**connect a **Widget** by holding IDs of the input and output **WidgetSockets** that have been configured.
* Comes with **Actions**(e.g. show an image when property changes) and **Editors** (e.g. displaying radio buttons for boolean property) for use in widgets.
* Actions and Editors could not be made **socket-aware** until 5.1.
* Widgets Parts

widget type

id

name

view

ControllerClass

* UI config:
  + **cockpit-config.xml -**The merged **UI** configuration of the entire backoffice application. Modified from AO.
  + **myextension/resources/myextension-backoffice-config.xml -**UI configuration of cockpit components at the extension level.
  + **myextension/backoffice/resources/widgets/mywidget/cockpit-config.xml -**Optional UI Widget configuration. Can be considered as a default configuration.
* Application config:
* defines services needed by backoffice components
* Widgets are defined in **myextension/backoffice/resources/widgets/mywidget/definition.xml** following widget-definition.xsd. Root elem with **unique** ID <widget-definition id="org.myextension.widgets.mysearch"...
* Define sockets with <sockets]] ]]> <input type="java.lang.String" id="incomingMsg"/><output type="java.lang.String" id="outgoingMsg"/>...
* View file in **myextension/backoffice/resources/widgets/mywidget/view.zul**.
* Backoffice widgets can be restricted from view via **AuthorityGroups** (which are **Authorities/roles**). This is deduced from the groups the current session user exists in.
* AuthorityGroups are created from hMC, via Backoffice Role, and setting Authorities from the Administration tab.

Web Services

* **REST** Web Service API, providing secure **CRUD** operations on Hybris models as well as actions. Delivers JSON or XML.
* All items are automatically exposed to WS as R**esources**.
* Models are transformed via **GraphTransformer** (Service Layer) into DTOs that are **JAXB**:ed (Resource Layer) to the client.
* Resources are **JAX-RS** POJOs, e.g. CatalogsResource has a getAllCatalogs() called at GET http://....../catalogs URI.
* **DTOs** are POJOs annotated with JAX-B annotations to map to/from models.  
  Also annotated with GraphNode annotation so GraphTransformer knows which model to map to/from.
* To create a **custom WS extension**, use yempty and extgen, rebuild, then run ant **webservice\_nature**. Add custom resource beans to **/resources/<extension>-web-spring.xml**.
* Every REST call is secured via authentication and authorization. Permissions per resource and attribute level.
* Security is based on Spring Security and applied via strategies, either Access Manager (hMC), properties based, or custom.
* By default Access Manager is used, and only allows users in **webservicegroup** that is NOT created by default (name configured as property '**webservices.security.group**'=).
* A strategy is set in **platformwebservices-web-spring.xml** on the **securityStrategy** attribute of bean **AbstractResource**.
* Access Manager security thus follows hMC access rights, i.e. per item read/write on item and attribute level.
* Properties strategy in project.properties as ''[restjersey.security.de](http://restjersey.security.de/)=admingroup[GET,PUT,DELETE,POST]'.
* Custom strategy defined bean extending CustomSecurityStrategy, and implement SecurityStrategy.
* For performance, caching (ETag/strategies), model attribute selectors, pagination is possible. OR call a FlexibleSearch query directly.

Third Party Databases

* Supported 3rd party DBs: HSQLDB (pre-bundled), Oracle, MySql, MSSQL, Hana.
* HSQL only allows one connection (not for use in production environments!)
* Local.properties: db.url, db.driver, db.username, db.password

MySQL

* jdbc:mysql://localhost/hybris?useConfigs=maxPerformance - maxPerformance sets properties for optimal performance.
* Driver: **com.mysql.jdbc.Driver.**
* Set MySql database to UTF-8 encoding.
* Set**innodb\_flush\_log\_at\_trx\_commit=0** or performance suffers.
* InnoDB (transaction support) preferred over MyISAM (no transaction support).
* [Using MySQL](evernote://wiki.hybris.com/view/14342068/s124/4cf8f82f-fb71-4ff9-90ca-2846ae59d7e8/4cf8f82f-fb71-4ff9-90ca-2846ae59d7e8/)

Media

* **Media** can be anything that can be stored on the filesystem.

**MediaItem** is container object that holds reference to a file - not a file, a reference to the physical representation on the file system (**MediaFolder**).

- has an identifier

- is assigned to a catalog version

- has a URL that points to location of actual file (media.getURL())

* Can organize **MediaItems** into **MediaFolders** (which have Storage Strategies and URL Strategies)
* Formats and Conversions
  + hybris can scale, convert and modify image media using ImageMagick
  + **MediaFormats** are only tags that can be applied to Media, **MediaContainers** can be used to group the same MediaModels with different formats
* SecureMediaFilter can be added to your webapp filter chain to restrict media access (<https://wiki.hybris.com/display/release5/Secure+Media+Access>)
* Media Storage Plugins (Pluggable Media Storage, Pluggable Delivery Strategies) (<https://wiki.hybris.com/display/release5/Media+Storage+Overview>)
* MediaConversion (<https://wiki.hybris.com/display/release5/mediaconversion+Extension+-+Technical+Guide>)

Security / Restrictions

* User accounts can be individual people (jsmith) or roles (productmanager)
* Principal defines User and PrincipalGroup
* PrincipalGroup defines UserGroup & Company (Company can have an address, UserGroup cannot)
* User defines Employee & Customer
* anonymous, admin and admingroup cannot be removed!
* Most everything has a SessionContext in hybris - therefore a user! (hMC, Cockpits, Web Services, Order Process, Addresses, CronJobs, etc)
* Types can be restricted by access rights...entire type or attribute will be hidden from display to the user account
* Permissions can be checked via API calls (**PermissionCheckingService**, **PermissionCRUDService**)
* Restrictions define a filter that is added to FlexSearch statements at execution time...they have a system-wide effect!!
* Spring security is used in the cockpits and accelerators...delegates authentication / authorization almost seamlessly
  + Every web app needs its own configuration!!

Transactions

* hybris provides an implementation of the Spring PlatformTransactionManager (which allows for the "normal" Spring transaction setup/config)
* BUT...no JTA!
* Isolation level is fixed at **READ\_COMMITED** (cannot be changed!!)
* Models are persisted without transactions by default (**model.service.transactional.saves**=false)

Caching

* OOTB Region-based caching (> 5.0)...implementation of EHCache (but other 3rd party caching APIs can be used)
* Each cache can be configured by type, max size, eviction policy (FIFO, LFU, LRU)
* The default setup of the hybris Region Cache contains the following regions:
  + Type system region: For storing entities of type system items
  + Entity region: For storing entities of all types except type system ones
  + Query results region: For storing all query results
  + Media items region: For storing all media items
* NO master cache server (invalidation will invalidate on all servers)
  + Caches of all nodes communicate with each other using TCP or UDP communication protocol.
  + The following is an overview of the **cache invalidation process**:
    1. A description of a product has changed. Therefore, all cache entries referring to the product are invalid.
    2. A cluster node on which the modification has been done sends a notification to all cluster nodes that all cache entries holding the product are invalid.
    3. Nodes that hold the product in their cache discard the cached data of the product and re-retrieve the product from the database once the product is used the next time.
* When calling getters/flex search, the data is returned from the cache if possible
* **Calling modelService.save invalidates the cached value**

Validation

* hybris' validation is based on the standard JSR 303 spec - easy and extensible way to validate data
* Allows for meaningful notifications to users
* Constraint defs are written in Java and XML - then placed into the DB as items
* Can have custom constraint types
* Data Validation consists of 3 areas
  + **ValidationService** in the service layer which defines constraints and performs data validation
    - loads validation engine with constraints
  + Administration cockpit which allows you to create / manage constraints
  + Cockpit integration to provide feedback to users

WCMS



References

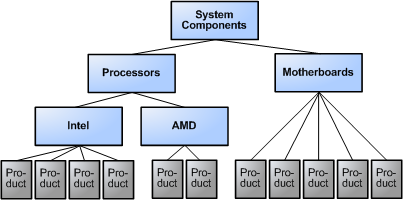
* + The product item references a catalogVersion item, which is identified using two

keys: a catalog reference and a version string. The catalog reference, in turn, is

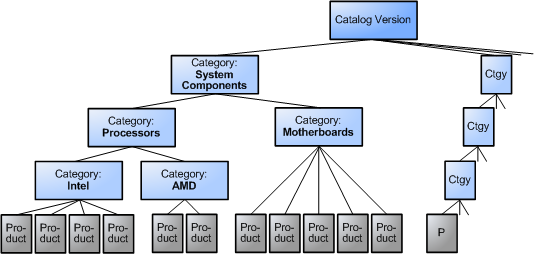
identified by an id string.



# Creating and Nesting Categories



# Hold Your Category Structure in Catalog Versions



Both a hybris catalog version and a typical printed product catalog, for example, represent your offered collection of products at a certain point in time. The differences between the two is that a printed product catalog is just a list of sequenced products. In hybris catalog versions, you can organize products in hierarchical category structures. It allows automated content modification. It enables you to maintain more and very specific information types, for example metadata, hidden product data, localizations, views, and restrictions. Instead of paper only, you can propagate your products in multiple channels like web sites, electronic catalogs, mobile, point of sales terminals, or tele-commerce applications. To enable channel specific propagation, the layout is defined separately from the content.

For example, while you can have one catalog version for editing the content (**Staged**), you can use another catalog version for propagation as a web shop (**Online**). Thus, a catalog is not only a container for holding and structuring your products but also a structure for basic maintenance and propagation processes.

**S0019820707**

**Username：alanliuxiang**

**Password: Lxiang77**

