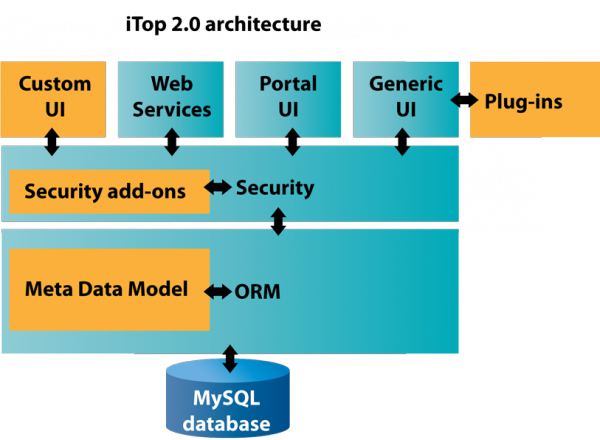
**iTop Customization**

|  |
| --- |
| **This document applies to iTop version 2.0. For customizing previous versions of iTop refer to** [**the old toolkit**](https://wiki.openitop.org/doku.php?id=toolkit_for_modifying_the_data_model)**. The 2.0 toolkit is available here:** [**iTopDataModelToolkit-2.0.zip**](http://www.combodo.com/documentation/iTopDataModelToolkit-2.0.zip) |

iTop is built on top of an ORM (Object Relational Mapper) abstraction layer that relies on the definition of a “Meta Data Model” made of PHP classes. Starting with iTop 2.0 the Meta Data Model can be described in XML which is then “compiled” into PHP classes during the setup of the application.

The iTop architecture can be depicted as the schema below:

[](https://wiki.openitop.org/lib/exe/fetch.php?media=2_1_0:customization:itop-architecture.png)

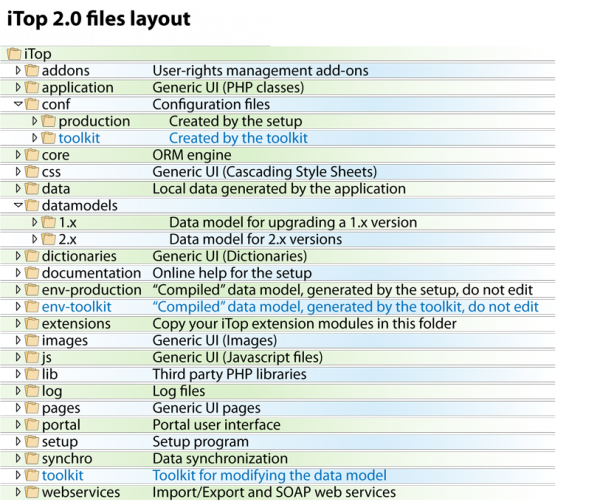
The orange boxes on the schema above are the parts of the application that can be customized.

| **Type of customization** | **Mean** | **Documentation** | **Intended audience** |
| --- | --- | --- | --- |
| Extending or modifying the meta data model | XML files | [XML reference](https://wiki.openitop.org/doku.php?id=2_1_0:customization:xml_reference) | iTop consultants, ITIL specialists |
| Extending the generic user-interface via a plug-in | PHP files | [Extensions API](https://wiki.openitop.org/doku.php?id=2_1_0:customization:extensions_api) | PHP developers |
| Creating new user interface pages to implement new web services, specialized exports or a specific task oriented user interface | PHP files | ORM API | PHP developers |
| Creating your own security scheme | PHP files | User Rights API | PHP developers |

Starting with iTop 2.0, the modification of the meta data model can be done by writing a simple XML file that will contain only the differences with the standard data model. By doing so, your modifications are kept in a separate file, that will remain applicable to the next version of iTop.Thus preserving your customizations in case of upgrade.

**Understanding the iTop file structure**

The layout of the iTop files and folders can be depicted as below:

[](https://wiki.openitop.org/lib/exe/fetch.php?media=2_1_0:customization:itop-files-layout.png)

The following folders have a special usage in iTop:

* **conf**: contains the configuration files, with one sub-folder per environment (see Environments below).
* **data**: contains application generated data, like the image for the object life-cycle (if graphviz is available on the system)
* **datamodels**: contains the meta data model definitions, with one sub-folder per major version of iTop.
* **env-xxxx**: these folders (one per environment) contain the “compiled ” data model. The env-production folder is completely re-created each time the application is re-installed or upgraded. If you edit its content, be aware that your modifications *will be lost* upon re-install or upgrade.
* **extensions**: this folder is the place where to copy extensions modules that are not part of the standard distribution of iTop.
* **log**: contains the log files of the application: setup.log and error.log

All the other folders should be considered as part of the source code of the application and should generally not be modified. The application never writes to these folders, they can be marked as read-only for the web server process.

**Environments**

A new concept introduced by iTop 2.0 is the notion of “environment”. An environment is an instance of iTop running the same code base but with potentially its own data model and configuration file. An environment is made of:

* A configuration file stored in conf/name\_of\_the\_environment/config-itop.php.
* A data model runtime, stored in env-name\_of\_the\_environment

Two different environments can operate on the same database if their configuration files says so.

The toolkit automatically creates a separate environment (named toolkit) in order to compile the XML data model and test its consistency without affecting the “production” environment. When the changes are Ok, you can instruct the toolkit to apply the validated changes to the “production” environment.

**Extension Modules**

The basic unit of data model definition in iTop is called a module.

A module groups together all the files needed to deliver a given feature: data model definitions in XML, PHP classes, Javascript and CSS files, PHP pages, images, etc… A module contains at least one file: the module definition file, always named module.name\_of\_the\_module.php.

Though you can always patch the source code, the best way to customize iTop consists in writing your own module. This creates a clean packaging of the customization and allow an easy (re)installation for your deployment or in case of upgrade.

**PHP versus XML data model definitions**

In iTop versions 1.x, the data model definition was written as a plain PHP classes. iTop version 2.0 support both PHP and XML definitions for the data model.

XML definitions have one major advantage over PHP definitions: the XML definition in one module can alter the data model defined in another module. For example, it is possible to create an extension module that will - when installed - add an attribute to the standard class “Server”, which is defined in the standard iTop data model. The extension module does not need to replace the whole “itop-config-management” module (where the Server class is defined), it can just alter the definition of a classe defines elsewhere.

In order to achieve the same effect in PHP, the only solution consists in cloning the whole itop-config-mgmt module and replacing it with your own patched version. When a new version of the module is released, you have to redo this diff & patch work again to produce a new version of your own module. Since an XML definition is directly defined as a difference, the upgrade is automatic.

**Content of a module**

If *my-module* is the name of your module, the module folder will contain the following files:

| **File Name** | **Description** |
| --- | --- |
| module.*my-module*.php | The module definition file. Mandatory. Contains the description of the module (name version, dependencies on other modules, etc.) and its components. |
| datamodel.*my-module*.xml | Data Model in XML. Upon installation the “compilation” will produce the model.xxxxx.php file based on the XML definitions. The XML file can contain the definition of classes, menus and profiles |
| model.*my-module*.php | If you choose to define the data model directly in PHP, then this file is the place to put such definitions |
| main.*my-module*.php | PHP code and utilities. For some modules that contain a lot of PHP code, it is easier to extract the PHP code and edit it in this separate file than letting the code embedded in the XML. |
| images | It is a good practice to store the images (classes icons, etc…) in their own sub folder |

The module names starting with *itop-* are reserved for offical modules from the iTop package. To avoid naming conflicts with other extensions it is recommended to name your custom modules with a name starting with the name of your company. For example the custom modules developed by Combodo are named *combodo-xxxx*.

**Creating a module**

Fill the form below and click on “Generate” to generate an empty module as the starting point for your customization:

窗体顶端

|  |  |  |
| --- | --- | --- |
| **Module Name**: |  | (must be unique. Names starting with "itop-" and "combodo-" are reserved for use by Combodo) |
| **Module Label**: |  | (Displayed during the setup) |
| **Module Version**: |  |  |
| **Category**: |  |  |
| **Dependencies**: |  | (comma separated list of module names/versions) |
|  |  |  |
|  | | |
|  |  |  |

窗体底端

**Installing the Toolkit**

* Download the toolkit zip file: [iTopDataModelToolkit-2.0.zip](http://www.combodo.com/documentation/iTopDataModelToolkit-2.0.zip)
* Expand the content of the zip file to create a directory “toolkit” at the root of the your developement iTop instance.
* Point your browser to **http://<your\_itop>/toolkit**

**Development Workflow**

1. Create an empty module
2. Install a development instance of iTop, including your empty module in the “extensions” folder
3. Install the toolkit on your developement instance
4. Edit your extension module and validate it with the toolkit
5. Apply the changes made to your extension module to the “production” environment
6. Test your module with some sample data. In case of troubles, fix them by iterating at point 4.

XML files must be “compiled” to PHP files each time you modify them, this is the job of the toolkit. However all other files in your module are simply copied from the “extension/your\_module” folder to “env-production/your\_module” folder. To speed-up the debugging on Linux systems you can replace these files by a symbolic link to their actual source file; Thus any modification to the source is immediately effective in iTop, you just have to hit the refresh button of the browser.

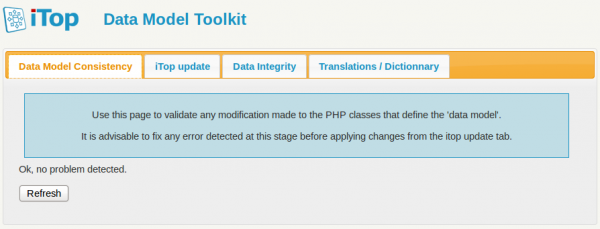
When your extension is completed you can deploy it on your production system by:

1. Copying the folder containing your extension module to the “extensions” folder on the production system
2. Marking the configuration file as read/write
3. Running the setup again and selecting your module in the list of “extensions” at the end of the interactive setup

**Using the Toolkit**

Once the toolkit is installed, point your browser to: **http://<your\_itop>/toolkit**.

The first tab performs some basic consistency checks and validation of the data model definition. You can use the “Refresh” button to perform the validation again each time the data model definition has changed. The checks performed in this tab work on the specific “toolkit” environment and thus have no effect on the actual iTop instance that uses the “production” environment.

[](https://wiki.openitop.org/lib/exe/fetch.php?media=2_1_0:customization:toolkit1.png)

The order in which the XML files are loaded is important (especially if an XML file alters the definitions given in another one). This loading order follows the “dependencies” declared in the module description file.

If you get an error like:

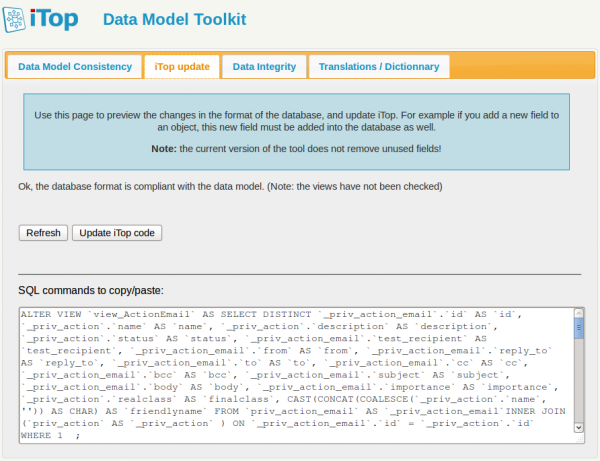
XML datamodel loader: could not find node for class/XXXX

This probably means that your module lacks a dependency on the module where the class XXXX is defined.

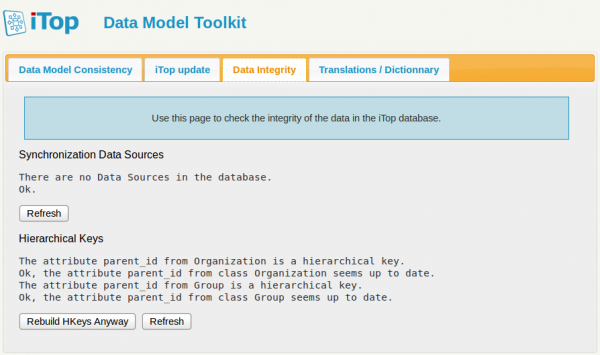
When the setup runs the loading order is computed and the resulting order is written in the configuration file. Be aware that the toolkit does *not* recompute this loading order. Therefore if you miss a dependency, you need to modify the module definition file to add this dependency and run the setup again in order to recompute the whole configuration file

When the first tab does not show any error, you can move to the second tab to:

* check the change to the database schema
* apply the changes of the database schema and data model definitions to the “production” iTop environment by clicking on “Udpate iTop Code”

[](https://wiki.openitop.org/lib/exe/fetch.php?media=2_1_0:customization:toolkit2.png)

The third tab of the toolkit can be used to update the Data Synchronization sources (if there are any that was impacted by the changes to the data model), and to check if there are any discrepencies in the internal data maintained for hierachical keys. This tab directly operates on the “production” environment.

[](https://wiki.openitop.org/lib/exe/fetch.php?media=2_1_0:customization:toolkit3.png)

**Life-cycle images**

A graphical representation of the object's life-cycle is displayed in the “Data Model” page of iTop (tab “Life-cycle”). If you modify an object's life-cycle, then you'll need to have Graphviz installed on the web server to get this images re-computed. See [Installing iTop](https://wiki.openitop.org/doku.php?id=2_1_0:admin:installing_itop#optional_dependencygraphviz) for more information about installing Graphviz.