



# OpenSpecs Technical Documentation

<b>General information.....</b>	<b>2</b>
<b>Logical structure of OpenSpecs.....</b>	<b>3</b>
<b>Folders structure of OpenSpecs .....</b>	<b>4</b>
<b>Implementation of content types at a program level.....</b>	<b>5</b>
<b>Implementation of installation procedure.....</b>	<b>6</b>
<b>Implementation of unique items numbering .....</b>	<b>7</b>
<b>Tools description .....</b>	<b>8</b>
<b>Project tree.....</b>	<b>8</b>
<b>Analyzer of architectural objects.....</b>	<b>8</b>
<b>Relations checker.....</b>	<b>8</b>
<b>Requirements and specifications document generator.....</b>	<b>8</b>
<b>Concepts editor.....</b>	<b>8</b>
<b>Mapping implementation.....</b>	<b>9</b>
<b>Discussion organization.....</b>	<b>Error! Bookmark not defined.</b>
<b>OpenSpecs versions history .....</b>	<b>10</b>
<b>OpenSpecs 1.00 version.....</b>	<b>10</b>
<b>OpenSpecs 0.6 beta final version .....</b>	<b>10</b>
<b>OpenSpecs 0.5.2 beta version.....</b>	<b>11</b>
<b>OpenSpecs 0.5.1 beta version.....</b>	<b>11</b>
<b>OpenSpecs 0.5 beta version.....</b>	<b>11</b>
<b>OpenSpecs 0.4 beta version.....</b>	<b>12</b>
<b>OpenSpecs 0.3 beta version.....</b>	<b>12</b>
<b>OpenSpecs 0.2 beta version.....</b>	<b>12</b>
<b>OpenSpecs 0.1 beta version.....</b>	<b>12</b>
<b>OpenSpecs 1.3 alpha version.....</b>	<b>12</b>
<b>OpenSpecs 1.2 alpha version.....</b>	<b>13</b>
<b>OpenSpecs 1.1 alpha version.....</b>	<b>13</b>
<b>OpenSpecs 1.0 alpha version.....</b>	<b>14</b>
<b>OpenSpecs 0.9 alpha version.....</b>	<b>14</b>
<b>OpenSpecs 0.8 alpha version.....</b>	<b>14</b>
<b>OpenSpecs 0.7 alpha version.....</b>	<b>14</b>
<b>OpenSpecs 0.2 alpha version.....</b>	<b>14</b>
<b>OpenSpecs 0.1 alpha version.....</b>	<b>14</b>



## **General information**

### **OpenSpecs version 1.00.**

**Description:** OpenSpecs is a requirements and specifications capturing tool supporting a coherent and unified system development methodology based on interacting entities paradigm.

**Author:** Vitaliy Mezhujev (for Open License Society).

OpenSpecs is developed as a product of Plone Content Management System. OpenSpecs is tested with Plone of version 2.1.1 (based on Archetypes 1.3.5-final and Zope 2.7.8-final).

OpenSpecs implements the Systems grammar, which defines structure of object oriented data base of Zope (ZODB). Last version of implemented in OpenSpecs Systems Grammar is 1.1.6.

### **Plone products needed for OpenSpecs right functioning:**

ATSchemaEditorNG (OpenSpecs tested with version 0.3.2).

FCKeditor (OpenSpecs tested with version 2.0.2).

### **Programming languages used:**

1. PythonWin 2.3.5 (#62, Feb 8 2005, 16:23:02) [MSC v.1200 32 bit (Intel)] on win32.
2. Zope Page Templates as an HTML/XML generation tool. Used Tag Attribute Language (TAL) and Macro Expansion TAL (METAL).



## Logical structure of OpenSpecs

Each concept of the Systems Grammar implemented as a separate content type: Project, ReqSpecs, Requirement, Specification, NormalCase, TestCase, FaultCase, Issue, Architecture, Entity, Interaction, Attribute, Function, Interface, WorkPlan, WorkPackage, ChangeRequest, Task.

Each content type is implemented with python language and saved in corresponding its name file in OpenSpecs folder with py extension.

Logical hierarchy of content types:

Root of Plone site

Project

ReqSpecs

Requirement

Specification

NormalCase

TestCase

FaultCase

Issue

Architecture

Entity

Attribute

Function

Interface

Interaction

WorkPlan

WorkPackage

ChangeRequest

Task



## Folders structure of OpenSpecs

**OpenSpecs** // Main product folder (in \Plone 2\Data\Products\)

**Extensions** // Extensions folder

Install.py // Installation procedure

**skins** // Skins folder

**OpenSpecs** // OpenSpecs skins folder

// Icons of content types

Architecture.gif, Attribute.gif, ChangeRequest.gif, Entity.gif,  
FaultCase.gif, Function.gif, Interaction.gif, Interface.gif, Issue.gif,  
NormalCase.gif, Project.gif, ReqSpecs.gif, Requirement.gif,  
Specification.gif, Task.gif, TestCase.gif, WorkPackage.gif,  
WorkPlan.gif

Check.pt // Check page template

ManageProjectTree.pt // Manage Project Tree page template

ProjectTree.pt // Project Tree page template

Query.pt // Query page template

results.pt // Query results page template

editToolProperties.py // Macros for ManageProjectTree

tool.gif // Icon for OpenSpecs tool

\_\_init\_\_.py // Initialization file which import content types and tools

config.py // Definitions of internal project name and content  
management permissions

Graph.py // Graphical tool (is in the process of development)

// Python files with OpenSpecs content types definitions

Architecture.py, Attribute.py, ChangeRequest.py, Entity.py, FaultCase.py,  
Function.py, Interaction.py, Interface.py, Issue.py, NormalCase.py,  
OpenSpecsTools.py, Project.py, ReqSpecs.py, Requirement.py,  
Specification.py, Task.py, TestCase.py, WorkPackage.py, WorkPlan.py

History.txt // OpenSpecs development history

ReadMe.txt // Description of OpenSpecs installation procedure

refresh.txt // File allowing refreshing a product without Zope  
restarting

version.txt // Text file containing OpenSpecs version



## Implementation of content types at a program level

Any OpenSpecs content type is implemented as **python class**.

All OpenSpecs content types are inherited from two Plone basic classes:

**BaseContent** and **OrderedBaseFolder**.

**BaseContent** – the class which inherits properties and behaviour of a content item.

**OrderedBaseFolder** – is the allowing ordering container for other content items.

OpenSpecs uses **archetypes** as a framework for developing new content types in Plone. The heart of an archetype is its **schema**, which is a sequence of fields.

The schema is a definition of what a content item will contain and how to present the contained information.

OpenSpecs uses **BaseSchema** which includes two fields: **id** and **title** (and also standard metadata fields). A schema of any OpenSpecs content type is formed from BaseSchema with adding to it concrete fields.

In OpenSpecs fields are used:

**StringField**, **TextField**, **LinesField**, **BooleanField**, **DateTimeField**,  
**ReferenceField**.

For information visualisation the widgets are used:

**StringWidget** (for string entering, e.g. Title).

**RichWidget** (enables the currently selected WYSIWYG Visual Editor like FCKEditor).

**SelectionWidget**, **MultiSelectionWidget** is UI widgets that let you choose one or many choices.

**ReferenceWidget** (implements referencing).

**BooleanWidget** (implement boolean choice e.g. MetTestCase).

**CalendarWidget** (date choice e.g. Deadline).

In each class definition fixed **factory type information** allowing:

1. For folderish types filtration of content types (boolean attribute `filter_content_types`) and setting allowed content types (list `list_allowed_content_types` with content type names).
2. For all content types allowing discussion (`allow_discussion` boolean attribute) and setting content icon in gif format (`content_icon` attribute).



## Implementation of installation procedure

Installation procedure is implemented in Install.py file in OpenSpecs\Extensions folder. Main procedure in Install.py is **install** which:

1. Installs OpenSpecs types.
2. Installs OpenSpecs skins and registers them.
3. Installs OpenSpecs tools in the Plone site.
4. Adds configlet Manage Project Tree.
5. Installs catalog indexes.
6. Adds actions (ProjectTree, Query, Check, RSDocument, Concepts Editor).
7. Uninstalls calendar portlet.
8. Deletes top of navigation tree (to see only a project structure).
9. In the root of Plone site creates the Project, the Background documents, the Reference docs, the Meeting notes and the Standards folders.
10. Implements filtration of folderish types and root of Plone site:
  - 10.1. In the root of Plone site user can add File, Folder, Image and Project items.
  - 10.2. In a Folder in the root user can add File, Folder and Image items.
  - 10.3. In any OpenSpecs folderish type user can add File and Image items (to illustrate a content item).
11. Sets rights of members ('OpenSpecs: Add Content', 'Add portal content', 'Modify portal content', 'ATContentTypes: Add Image', 'ATContentTypes: Add Folder', 'ATContentTypes: Add File', 'ATContentTypes: Add Document', 'List undoable changes', 'Undo changes').

Also in Install.py implemented **uninstall** procedure which removes portal actions.



## Implementation of unique items numbering

In each OpenSpecs folderish content type the function `invokeFactory` is redefined, which forms items **identifier** (id) from first letters of content type name plus item unique number.

To generate unique number for each content type in `OpenSpecsTools.py` is defined set of functions: `getNumReqSpecs`, `getNumArchitecture`, `getNumWorkPlan`, `getNumRequirement`, `getNumSpecification`, `getNumNormalCase`, `getNumTestCase`, `getNumFaultCase`, `getNumIssue`, `getNumEntity`, `getNumInteraction`, `getNumAttribute`, `getNumFunction`, `getNumInterface`, `getNumWorkPackage`, `getNumChangeRequest`, `getNumTask`.

Unique items numbers is stored in ZODB and zeroed only after clearing ZODB catalog (or OpenSpecs reinstallation – its zeroing implemented in class constructor).

## Tools description

### ***Project tree***

Reflects hierarchical structure of OpenSpecs content types.

Implemented in ProjectTree page template with Tag Attribute Language (TAL) and Macro Expansion TAL (METAL).

Page template finds in portal catalog all allowing content items (with declared in OpenSpecsTools.py getObjectValues procedure).

All items shows as hyperlinks which allows quick moving to selected item.

### ***Analyzer of architectural objects***

Show all Requirements, Specifications, Normal, Test, Fault cases, Issues and Tasks for selected architectural object. For Interaction shows Entities participate in it.

Implemented in Check page template with TAL and METAL.

### ***Relations checker***

Implements formalized analyses in:

1. Requirements view – show Specifications which do not correspond to Test Cases (used Boolean field in Specification archetype).
2. Architectural view – show Entities and Interactions which you have to define. Used comparison of nominal model (i.e. vocabulary of entities and interactions names) with already defined Entities and Interactions.
3. Project planning view: show the tasks you have to define (used comparison of already define tasks and set of Entities and Interactions).

Implemented in Query and results page templates with TAL and METAL.

### ***Requirements and specifications document generator***

Generates output file in HTML format and saves all formatting which user have done in Rich Widgets (text structuring, images etc.). RS Document generator also allows work with dynamically changing content types.

### ***Concepts editor***

Implements possibility of managing (i.e. dynamic definition and deleting) fields of Systems grammar concepts through the web by user.

Concept editor uses ATSchemaEditorNG Plone product.

In Project.py in function manage\_afterAdd by the function atse\_registerObject implemented registration of content types which can be managed through Concept editor.



To allow editing schema each OpenSpecs folderish content type inherits from SchemaEditor class. Other classes inherit from ParentManagedSchema class.

### ***Mapping implementation***

Implement mapping between deferent (Requirements, Architectural, Work plan) views on project development. Uses special reference fields:

1. Into Requirement, Specification, Normal Case, Test Case, Fault Case, Issue and Task added architectural reference to identified Entities(s) and Interaction(s).
2. For feedback implementation in Entity and Interaction archetypes added references to appropriate Requirement(s) or/and Specification(s).
3. In specification user can specify the Reference to Requirement(s) of which the Specification is descendant.
4. In Specification added Boolean field 'MetTestCase' checking if Specification correspond to Test Case.

## OpenSpecs versions history

### ***OpenSpecs 1.00 version***

1. All OpenSpecs folderish content types are inherited from the OrderedBaseFolder class. So user now can reorder items in the content page. This arranging also influence on order of writing items in RSDoc.
2. Suppressed the error `atse_getDefaultSchemaId`. Now Concepts Editor tab pressing calls `atse_editor` from Project folder (it also means that in the root of Plone needed obligatory folder with Id Project).

### ***OpenSpecs 0.6 beta final version***

1. Concepts edition
  - 1.1. Requirement is transformed into folderish type and now can include as Requirements and Specifications items.
  - 1.2. Issues field is added in any content type (Note. Issue is also saved as a separate content type).
  - 1.3. Specification is moved from ReqSpecs into Requirement folder. It is implementation of indirect parent relations: Specification is considered as descendant of Requirement. We also have such kind of relations for Normal, Test and Fault Cases because Specification is their folder. For this reason Architectural refs and identifiers are deleted from Test, Fault and Normal Cases.
  - 1.4. Fault reasons field is added in Fault case.
  - 1.5. RS generator is modified to meet the new concepts structure. Requirements writing procedure is redeveloped into recursive to support enclosure of content types.
2. Installation procedure

At installation in the root of Plone site the Project, the Background documents, the Reference docs, the Meeting notes and the Standards folders are created.
3. Implemented filtration of folderish types and root of Plone site
  - 3.1. In the root of Plone site user can add File, Folder, Image and Project items.
  - 3.2. In a Folder in the root user can add File, Folder and Image items.
  - 3.3. In any OpenSpecs folderish type user can add File and Image items (to illustrate a content item).
4. Changing names and messages
  - 4.1. Requirements renamed into ReqSpecs.
  - 4.2. Site map renamed into Project tree.
  - 4.3. Analyses page renamed into Query.
  - 4.4. Help messages on the Check page are changed.
5. Unique numbering



Unique numbering for all content items of OpenSpecs is implemented. Ids are now readable and formed at creation from first letters of item type name + item unique number (before a Title assignment as a Title an Id is used).

Note. Unique items numbering is saved even after item deletion. To start new numbering user has to use Clear Catalog option.

#### 6. Interface improving

6.1. Status field in a content item showed first.

6.2. Requirement and specification signs on icons are changed to biggest.

6.3. Height of a visual text editor is decreased.

6.4. Top of navigation tree is excluded (now user can see only the project structure).

6.5. Calendar is deleted from standard Plone skin.

6.6. Add help (purpose) of analysis to Query page.

6.7. Add Project tree manage reference to Project tree web-page (Note. This option also selects the content types which will be written in RS Document).

6.8. HTML is set as the default content type of a visual text editor.

#### 7. Changing users rights

7.1. Manger of project has to confirm user registration (only manager can see prefs\_users\_overview web-page so add, delete, and change rights of users).

7.2. Concepts editor is only allowable for managers of project.

7.3. Project tree configuration page is allowable for any user of OpenSpecs.

7.4. At installation procedure rights of members are set ('OpenSpecs: Add Content', 'Add portal content', 'Modify portal content', 'ATContentTypes: Add Image', 'ATContentTypes: Add Folder', 'ATContentTypes: Add File', 'ATContentTypes: Add Document', 'List undoable changes', 'Undo changes').

### ***OpenSpecs 0.5.2 beta version***

1. RS Document generator now can work with images.

### ***OpenSpecs 0.5.1 beta version***

1. Limited possibility of deleting content types by users via Concepts Editor (user can manage only own fields). TBD. Bug with id field (user still can delete it).

### ***OpenSpecs 0.5 beta version***

1. Implemented RS Document generator which can work with dynamically changing content types.

### ***OpenSpecs 0.4 beta version***

1. Implemented possibility of managing (i.e. dynamic definition and deleting) fields of Systems grammar concepts through the web by user. Concepts Editor tool uses ATSchemaEditorNG 0.3.2 library.

### ***OpenSpecs 0.3 beta version***

1. All text fields of OpenSpecs now use Rich Widget allowing text structuring.  
2. Fully redeveloped RS Document generator. Now it generates output file in HTML format and saves all formatting which user has done in Rich Widgets.

### ***OpenSpecs 0.2 beta version***

1. Conceptual level of systems definition is renamed into Requirements.  
2. Added 'DescriptionText' field into Requirements and WorkPlan types (now each concept of systems grammar has at least one textual 'description' field).  
3. 'System' content type is deleted. Instead will be used 'Entity' folderish type. Entity can include other entities.  
4. Property content type is transformed into Attribute.  
5. Action content type is transformed into Function.  
6. Developed new content types: NormalCase, TestCase, FaultCase and Issue.  
7. Specification is transformed into a folderish type which can include NormalCase, TestCase, FaultCase and Issue.  
8. In each concept was added obligatory attribute - Status, which reflects progress in time of concepts development.  
9. Architectural reference from Requirement, Specification, Task, NormalCase, TestCase, FaultCase and Issue now can lead to 'Entity', 'Interaction', 'Attribute', 'Function', 'Interface' content types.  
10. Added 'Change Request' folderish content type (it can be included in Work Plan and contains Tasks).  
11. Site map now supports enclosure up to 5 levels.  
12. Modified RS Document Generator in conformity with renewed systems grammar. Also was implemented a multilevel numbering of document items.

### ***OpenSpecs 0.1 beta version***

1. Conceptual level of system definition is added (ReqSpecs contents type is renamed into Concepts).

### ***OpenSpecs 1.3 alpha version***

1. Added page templates of analyses methods.

2. From Object and Interaction contents types deleted reference fields to Requirement(s) and Specification(s), in exchange implemented algorithm of finding back (leading to object) references.

### ***OpenSpecs 1.2 alpha version***

1. Checked the method of RS document writing. From now RS document is generated from the Project level (for the first project in the site).
2. Added Purpose field in the Project contents type.
3. Implemented possibility of Architectural Referencing the System(s) at Requirement and Specification types.
4. Changed concept of Properties Types. In class Property added the method `get_voc` which add to the list of possible types all names of already identified objects. From now identified object can serve as a type of property (similar to OOP).
5. Changed name of field PropertyType to Property\_Type to avoid names conflict (`getPropertyType` caused mistake).
6. Action type (action invocation, message exchange) moved into Message archetype (Message has types: data exchange, action invocation).

### ***OpenSpecs 1.1 alpha version***

1. Mapping implementation:
  - 1.1. Into Requirement, Specification and Task archetype were added reference fields to identified Object(s) or Interaction(s).  
Field Name (of identified Object or Interaction) in Requirement, Specification was saved to collect vocabulary of objects names (i.e. build nominal model).
  - 1.2. For feedback implementation in Object and Interaction archetypes were added references to appropriate Requirement(s) or/and Specification(s).
  - 1.3. In specification we can specify the Reference to Requirement(s) of which the Specification is descendant.
  - 1.4. In Specification was added Boolean field 'MetTestCase' checking if Specification correspond to Test Case.
2. Automated procedure of contents types installation: you do not need now install each contents types separately.
3. Formalized checker implementation. The page 'check' is divided into.
  - 3.1. Requirements view – show Specifications which do not correspond to Test Cases (used Boolean field in Specification archetype).
  - 3.2. Architectural view – show Objects and Interactions which you have to define. Used comparison of nominal model (i.e. vocabulary of objects and interactions names) with already defined objects and Interactions.



3.3. Project planning view: show the tasks you have to define (used comparison of already define tasks and rest of Objects and Interactions).

4. Added History file.

### ***OpenSpecs 1.0 alpha version***

1. Installation procedure is simplified. Now you have sole OpenSpecs folder which includes both contents types and tools (Site Map, Requirements and Specifications Document Generator and Graph).

2. Added field 'Name' of identified object or interaction for Requirement, Specification and Task types (Title field is saved). To be in conformity with new fields R&S Document Generator also was changed.

3. At generation of R&S document types names are added.

4. Transformed into recursive the python procedure of Object definition, which allows definition any amount of subsystems.

5. Paragraph spaces between subsystems added.

6. Fixed bug with uninstallation of WriteRSDoc action.

7. Help on web-pages checked.

8. Added ReadMe.txt file with installation instruction.

### ***OpenSpecs 0.9 alpha version***

Added help on each page of OpenSpecs.

### ***OpenSpecs 0.8 alpha version***

Implemented requirement and specification document generator.

### ***OpenSpecs 0.7 alpha version***

Implemented Graph content type (OpenSpecs graphics visualizations).

### ***OpenSpecs 0.2 alpha version***

Developed OpenSpecs Site Map.

### ***OpenSpecs 0.1 alpha version***

Developed content types (Requirement, Specification, Object, Property, Action, Interaction, Interface, Task, ReqSpecs, System, WorkPlan, Project, WorkPackage).