

# openCRX Security

**Version 1.6.0**

**[www.opencrx.org](http://www.opencrx.org)**

**openCRX Security: Version 1.6.0**

by [www.opencrx.org](http://www.opencrx.org)

The contents of this file are subject to a BSD license (the "License"); you may not use this file except in compliance with the License.

You may obtain a copy of the License *here* (<http://www.opencrx.org/license.htm>)

# Table of Contents

<b>1. About this Book</b> .....	<b>1</b>
Who this book is for .....	1
What do you need to understand this book .....	1
<b>2. Prerequisites</b> .....	<b>2</b>
<b>3. Introduction to role-based security</b> .....	<b>3</b>
<b>4. Permissions</b> .....	<b>5</b>
Ownership Permissions.....	5
Security Example .....	7
Model Permissions .....	15
<b>5. Next Steps</b> .....	<b>16</b>
<b>A. Appendix</b> .....	<b>17</b>
<b>Bibliography</b> .....	<b>18</b>

# List of Figures

3-1. Role-based Security .....	3
3-2. Role-based Security Concept implemented by openCRX.....	4
4-1. UML Model SecureObject (Extract).....	5
4-2. System Attributes of an openCRX Object.....	6
4-3. Example Organization with 9 UserGroups and 9 Users.....	7
4-4. Example Data with 6 Objects .....	8
4-5. Access Permissions of User admin-Standard .....	9
4-6. Access Permissions of User head-Sales .....	10
4-7. Access Permissions of User salesrep1 .....	11
4-8. Access Permissions of User salesrep2 .....	12
4-9. Access Permissions of User salesrep3 .....	13
4-10. Access Permissions of User salesrep4.....	14
4-11. Access Permissions of Users head-Accounting, accountant1, and accountant2 .....	15

# Chapter 1. About this Book

This book describes the concept of role-based security as it is implemented by *openCRX*.

## Who this book is for

The intended audience are *openCRX* administrators.

## What do you need to understand this book

This book assumes a basic knowledge of security-related concepts and terms, including identity, authentication, and authorization. It is also helpful if you are familiar with *openCRX* accounts and user homepages (UserHomes).

## Chapter 2. Prerequisites

This book deals with concepts only, i.e. there are no prerequisites in terms of installing openCRX. It is helpful to have a basic knowledge of security-related concepts and terms, including identity, authentication, and authorization.

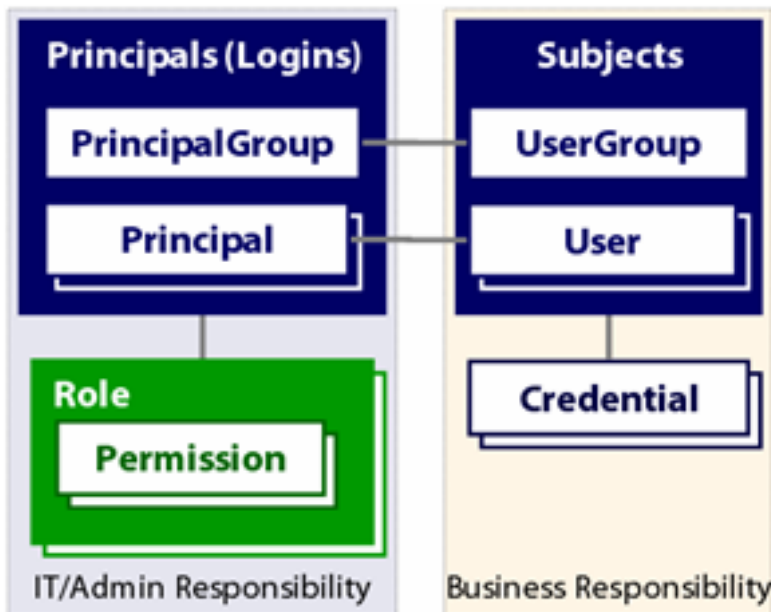
# Chapter 3. Introduction to role-based security

It is the main goal of the security concept of *openCRX* to **control who is allowed to browse/delete/update which information**. On the one hand this requires some infrastructure to manage **identity** data (e.g. principals, users), **authentication** data (e.g. passwords, certificates) and **authorization** data (e.g. permissions, roles), on the other hand the **rules defined by the various permissions granted to principals must be enforced** to ensure that no principal can take an action without being granted permission to do so. *openCRX* implements a state-of-the-art **role-based security concept** which utilizes the concepts of **Principals** and **Roles**, similar to the implementation of security in many current operating systems. While role-based security may be overkill in trivial settings (e.g. SOHO with two users who both are allowed to browse/delete/update all objects) it is an extremely powerful tool to get a handle on complex environments (e.g. ASP managing tens of thousands of customers/companies). Apart from these two extreme settings (SOHO, ASP), role-based security is also very useful in many typical company settings where various sales teams, customer service teams, and maybe accounting teams need to browse/delete/update customer-related data (e.g. sales orders, invoices, payments, support requests) while at the same time permissions on such data may vary depending on the function of the employee (imagine what would happen if accounting staff could enter/change sales orders or a customer service rep could alter customer payments).

The following explanations are hopefully helpful to understand the role-based security concept as it is implemented by *openCRX*. Please be aware that the goal of this documentation is to explain the concepts used in *openCRX* – if you are interested in details and the formally correct and complete specifications you should refer to the *openCRX UML models* (<http://www.opencrx.org/documents.htm#Doclatestuml>).

The following figure shows the main ingredients of role-based security:

Figure 3-1. Role-based Security

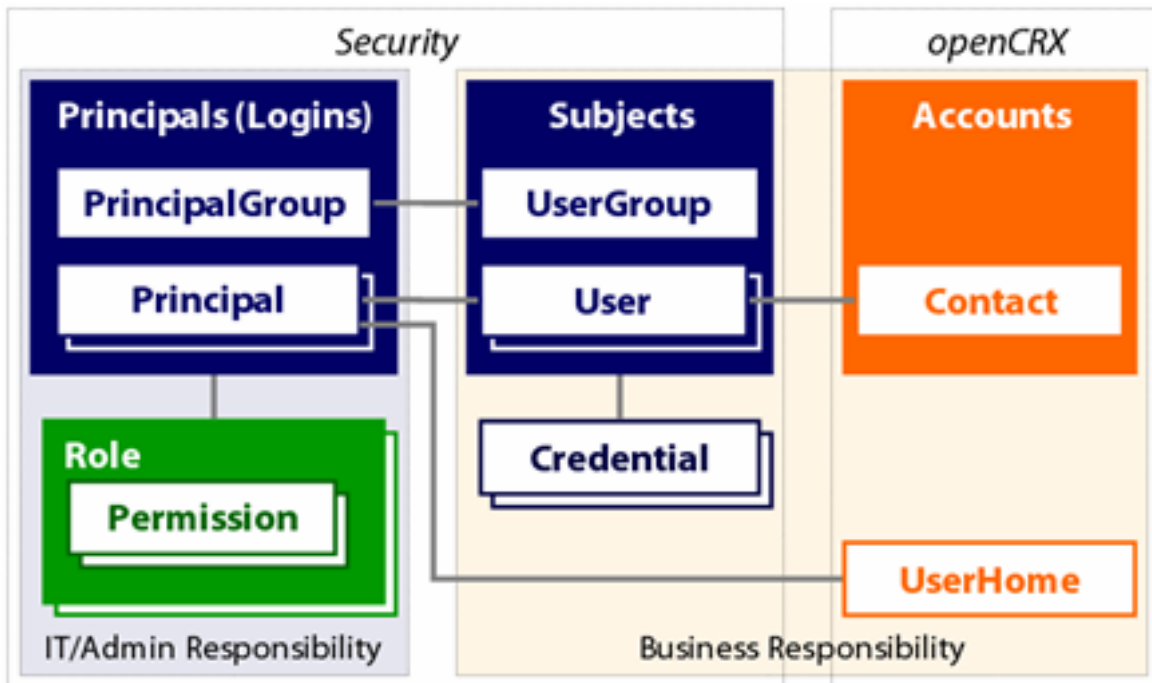


At the core of role-based security is the concept of collecting **Permissions** in **Roles**, which can be granted to **Principals** (a **Principal** corresponds to a **Login**). For example, imagine a **Principal** *admin-Standard* who is granted the **Role** *OpenCrxAdministrator* which comes with all the **Permissions** required for a segment administrator to do his work, or a **Principal** *salesrep1* who is granted the **Role** *SalesRep* which comes with all the **Permissions** required for a sales representative to do his work. For better manageability several **Principals** can be collected in a **PrincipalGroup** and **Roles** can be directly attached to **PrincipalGroups**.

**Users** (e.g. human beings like Joe, Mary, Jeff) can be assigned multiple **Principals** to reflect the fact that some users connect to the system in different roles depending on the tasks at hand. For example, **User joe** might be assigned the **Principal head-Sales** (because Joe is head of sales) as well as the **Principal admin-Standard** (because Joe is also segment administrator). If Joe wants to work as segment administrator he logs in as **Principal admin-Standard**, if Joe wants to work as head of sales he logs in as **Principal head-Sales**. **Credentials** (like passwords, certificates, SecurIDs) are attached to **Users**. Like this it is possible to let Joe connect to the system with the same password, regardless of whether he acts as segment administrator or head of accounting. For better manageability several **Users** can be collected in a **UserGroup**. The term **Subject** is commonly used to refer to **Users** and **UserGroups**.

In addition to this standard setup of role-based security openCRX allows you to relate **Users** to **Contacts**. Furthermore, each **Principal** is related (automatically) to an *openCRX* **UserHome** (this is how *openCRX* can easily maintain two different histories for **Principal admin-Standard** and **Principal head-Sales**; one history keeps track of Joe's actions when he is logged in as segment administrator, the other history keeps track of his actions when he is logged in as head of sales). These *openCRX* extensions to role-based security are shown in the following figure:

Figure 3-2. Role-based Security Concept implemented by openCRX



Please note that the above figure represents a high-level view of the implemented security concept – refer to the *openCRX UML models* (<http://www.opencrx.org/documents.htm#Doclatestuml>) for detailed and formally correct and complete specifications.

# Chapter 4. Permissions

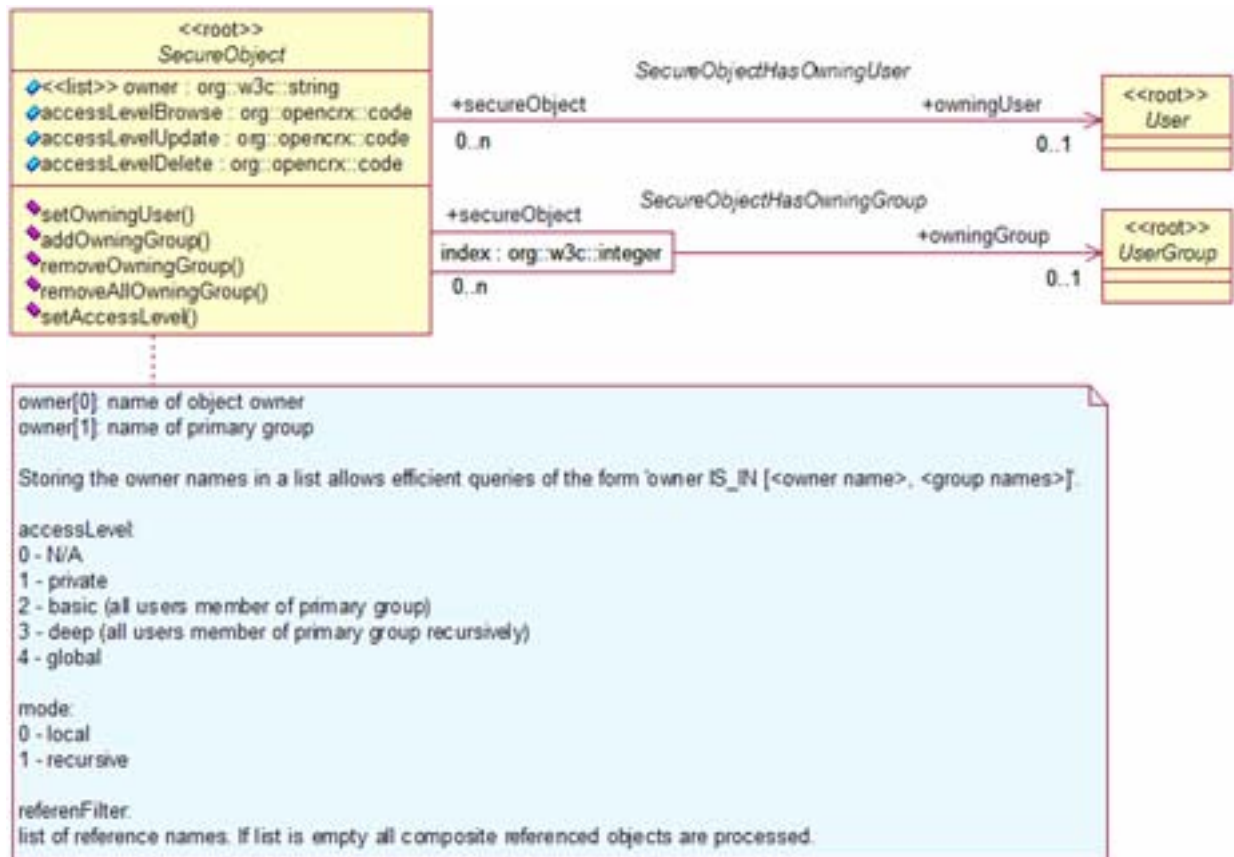
With the openCRX security framework permissions can be granted with the following two methods:

- *Ownership Permissions*: **Permissions are granted on a particular object based on ownership** (ownership access control has been implemented since openCRX v1.4.0). For example, permission to browse the account "Steve Jones" is granted to everybody or permission to delete the product "Gadget" is granted to the segment administrator only. Object permissions work similar to file permissions in a file system.
- *Model Permissions*: **Permissions are granted on model elements** (will be implemented in a future version of openCRX). For example, permission is granted to execute the operation **Refresh** on the UserHome or permission is granted to view the Tab **Leads**, etc.

## Ownership Permissions

Ownership permissions are used to control browse/delete/update access to openCRX objects by Users and UserGroups. **Ownership access control** has been implemented since openCRX v1.4.0. Every openCRX object is a *SecureObject*. The following figure shows an extract from the UML model (if you are interested in all the details and the formally correct and complete specifications you should refer to the *openCRX UML models* (<http://www.opencrx.org/documents.htm#Doclatestuml>)):

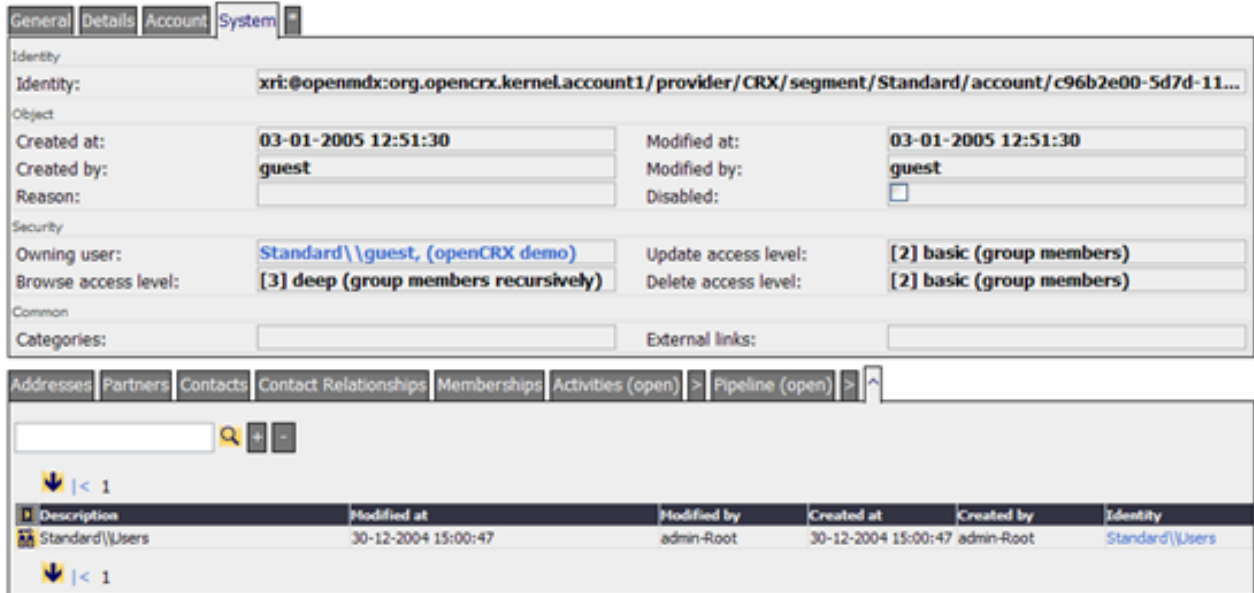
Figure 4-1. UML Model SecureObject (Extract)



The most important **security attributes** of an object X are discussed below (and an example is shown in *Figure 4-2*):

- **Owning User:** this user "owns" object X; the Owning User can always browse/delete/update object X (unless the access level is set to 0 [no access]).
- **Owning Groups:** these groups might enjoy privileged treatment for browsing/deleting/updating object X depending on the relevant access level settings.
- **Browse Access Level:** this setting determines which users / user groups are granted browse access to **direct composite objects of object X** [i.e. can **view/inspect** direct composite objects (including all their attributes) of object X].
- **Delete Access Level:** this setting determines which users / user groups are granted delete access to **object X and all its composite objects (recursively!)** [i.e. can **delete** object X and all its composite objects (recursively!)].
- **Update Access Level:** this setting determines which users / user groups are granted update access to **object X** [i.e. can **change** object X; this includes **adding composite objects to object X**].

Figure 4-2. System Attributes of an openCRX Object



The following **access levels** are available to control which users / user groups are granted permission to **browse direct composite objects of a particular object X** or **delete/update a particular object X**:

- **0 – N/A:** no access.
- **1 – private:** access is granted if the user is owning user of object X.
- **2 – basic:** access is granted if (a) the user is owning user of object X, or if (b) the user is member of any of the owning groups of object X, or if (c) any of the owning groups of object X is a subgroup\*\* of any group the user is member of.
- **3 – deep:** access is granted if (a) the user is owning user of object X, or if (b) the user is member of any of the owning groups of object X, or if (c) any of the owning groups of object X is a subgroup\*\* of any group the user is member of, or if (d) any of the owning groups of object X is a subgroup\*\* of any supergroup\* of any group the user is member of.
- **4 – global:** all users are granted access.

\* a supergroup of an owning group G is a group of which G is a member (recursively)

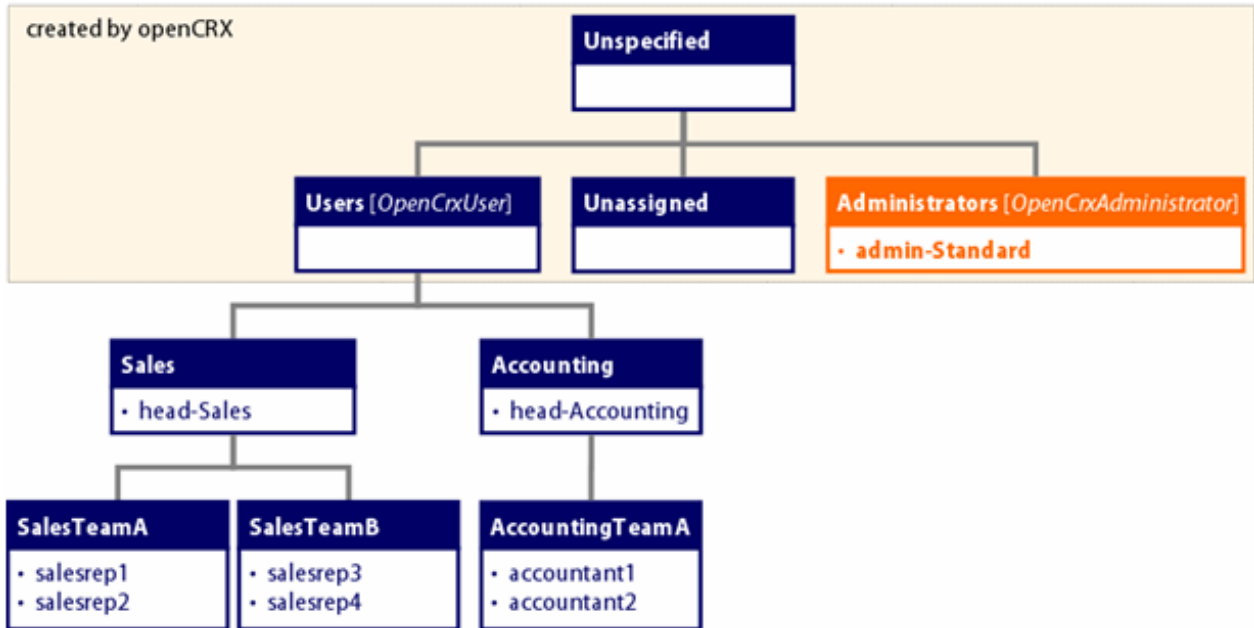
\*\* a subgroup of an owning group G is a group which is member of G (recursively)

## Security Example

In this chapter we discuss security for an example organization (see *Figure 4-3*) and then show what individual users are allowed to do on example data (see *Figure 4-4*).

The example organization consists of 4 User Groups (*Administrators*, *Users*, *Unspecified*, and *Unassigned*) and 1 User (*admin-Standard*) created by *openCRX*. In addition, there are 5 User Groups (*Sales*, *SalesTeamA*, *SalesTeamB*, *Accounting*, and *AccountingTeamA*) and 8 Users (*head-Sales*, *salesrep1*, *salesrep2*, *salesrep3*, *salesrep4*, *head-Accounting*, *accountant1*, *accountant2*) created by the segment administrator:

**Figure 4-3. Example Organization with 9 UserGroups and 9 Users**

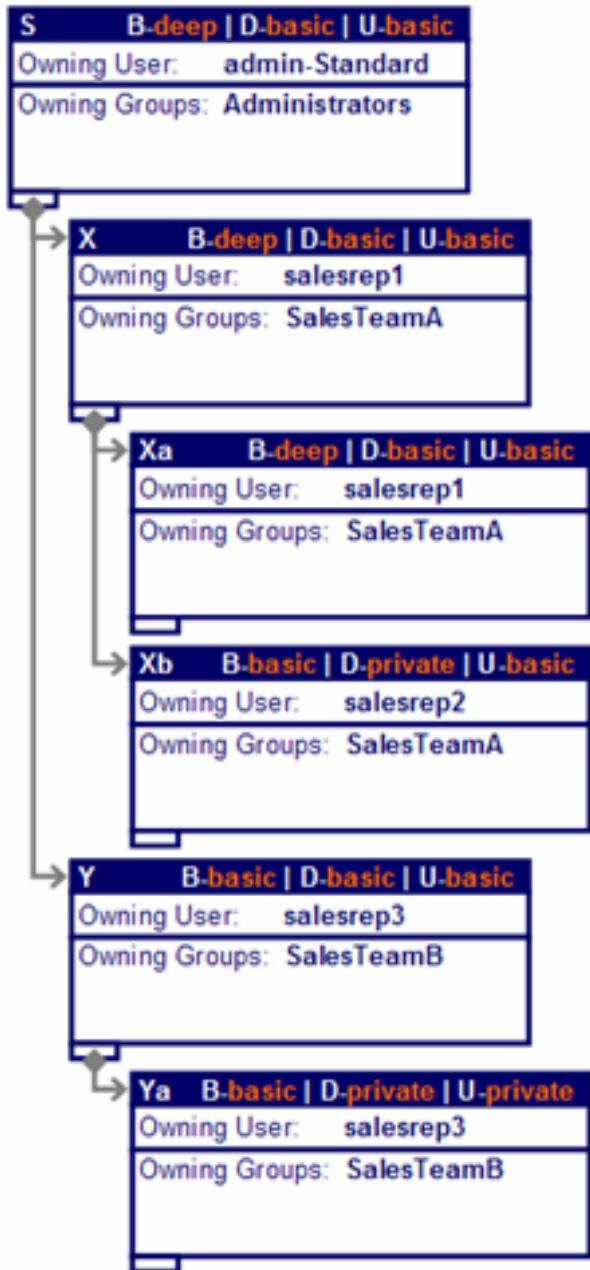


The above chart reads as follows (just to give a few examples, i.e. many more statements could be made about the above chart):

- **UserGroup** *SalesTeamA* is a subgroup of **UserGroup** *Sales*
- **UserGroup** *SalesTeamB* is a subgroup of **UserGroup** *Sales*
- **UserGroup** *AccountingTeamA* is a subgroup of **UserGroup** *Accounting*
- **UserGroups** *Sales*, *SalesTeamA*, *SalesTeamB*, *Accounting*, and *AccountingTeamA* are subgroups of **UserGroup** *Users*
- **UserGroup** *Users* is a supergroup of **UserGroups** *SalesTeamA* (**UserGroup** *Users* is also a supergroup of **UserGroups** *Sales*, *SalesTeamB*, *Accounting*, and *AccountingTeamA*)
- **UserGroup** *Unspecified* is a supergroup of all other **UserGroups**
- **User** *admin-Standard* is a member of **UserGroup** *Administrators*
- **Users** *salesrep1* and *salesrep2* are both members of **UserGroup** *SalesTeamA*
- **Role** *OpenCrxUser* is assigned to **UserGroup** *Users*
- **Role** *OpenCrxAdministrator* is assigned to **UserGroup** *Administrators*
- **User** *admin-Standard* has all the permissions of the **Role** *OpenCrxAdministrator* (because this **User** is a member of the **UserGroup** *Administrators*)
- ...

The example data consists of 6 openCRX objects (*S*, *X*, *Xa*, *Xb*, *Y*, and *Ya*):

Figure 4-4. Example Data with 6 Objects



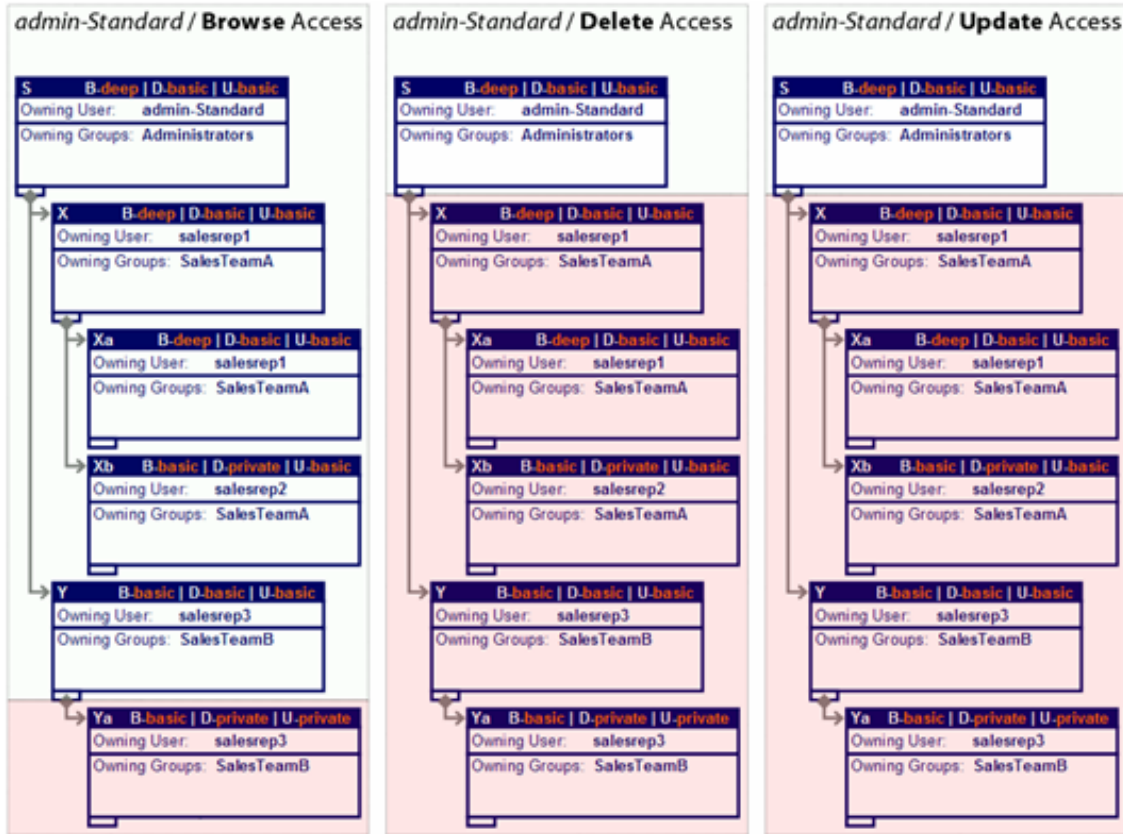
If you, for example, look at object *Xa* the following security attributes are set:

- **Owning User:** object *Xa* is owned by the User *salesrep1*
- **Owning Groups:** object *Xa* is owned by the UserGroup *SalesTeamA*
- **Browse Access Level** is set to *deep*
- **Delete Access Level** is set to *basic*
- **Update Access Level** is set to *basic*

Similarly, *Figure 4-4* contains information about the security attributes of all the other openCRX objects.

Next, we will analyze for a few of the example users which objects they are able to **browse** to, which objects they are allowed to **delete**, and which objects they have permission to **update**. The whole discussion is based on the security organization as it was presented in *Figure 4-3*. Let's start with the user *admin-Standard*. On the following chart, green indicates that access is granted, red indicates that access is not granted:

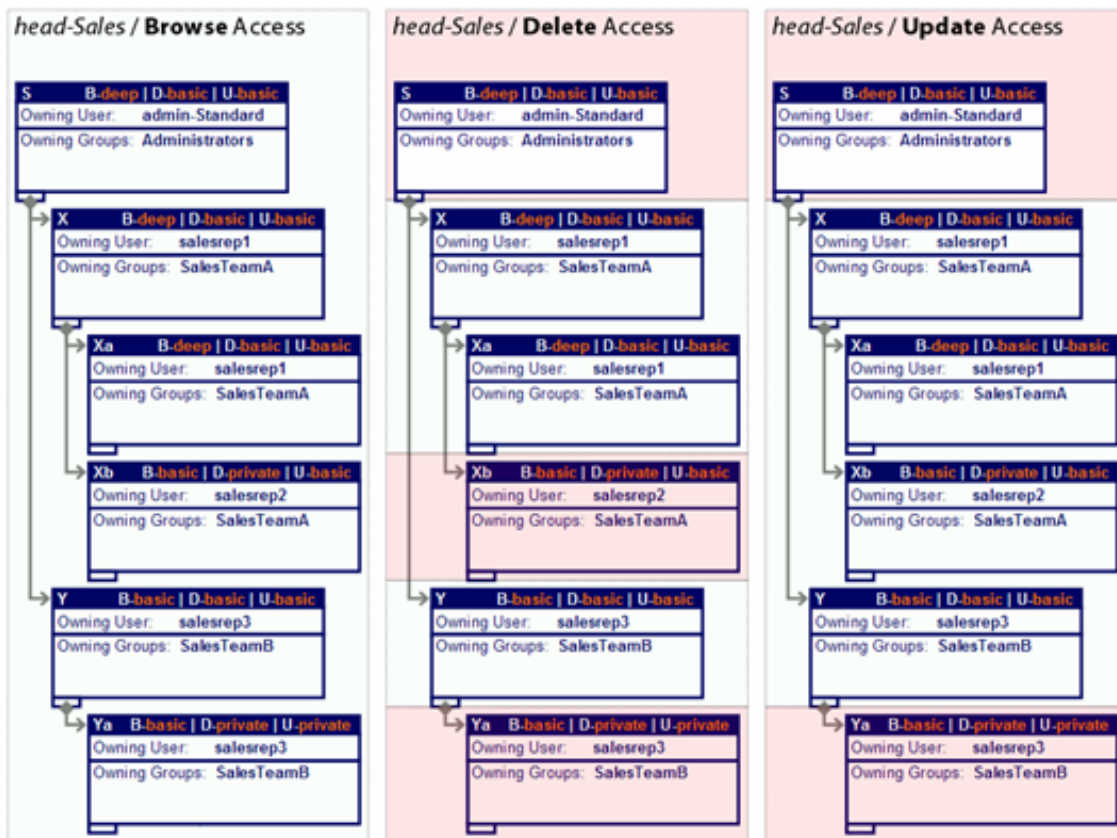
**Figure 4-5. Access Permissions of User admin-Standard**



**Warning** Please note that the user *admin-Standard* is granted permission to delete object *S*; due to the definition of the delete access level, deleting object *S* will also delete its composite objects (recursively!), i.e. *admin-Standard* can actually **delete all objects by deleting object S!** It is not possible, however, that user *admin-Standard* navigates to – for example – object *Xb* and then deletes object *Xb* as delete access is not granted on object *Xb* (even though browse level access to object *Xb* is granted).

Let's now look at the user *head-Sales*. On the following chart, green indicates that access is granted, red indicates that access is not granted:

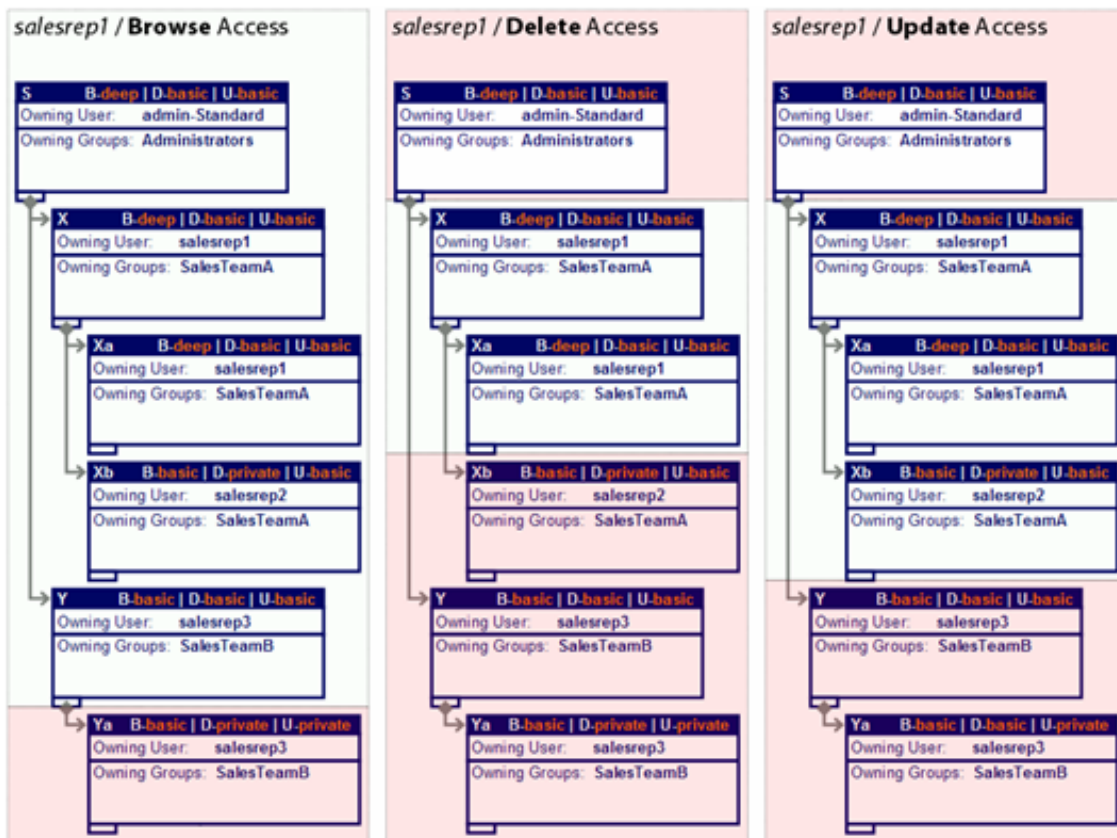
Figure 4-6. Access Permissions of User *head-Sales*



**Warning** Please note that the user *head-Sales* is granted permission to delete objects *X* and *Y*; due to the definition of the delete access level deleting object *X* will also delete its composite objects (recursively!), i.e. *head-Sales* can actually **delete object *Xb* by deleting object *X***! Similarly, *head-Sales* can delete object *Ya* by deleting object *Y*. It is not possible, however, that user *head-Sales* navigates to – for example – object *Xb* and then deletes object *Xb* as delete access is not granted on object *Xb* (even though browsing to object *Xb* is possible).

On the following chart, green indicates that access is granted to the user *salesrep1*, red indicates that access is not granted to *salesrep1*:

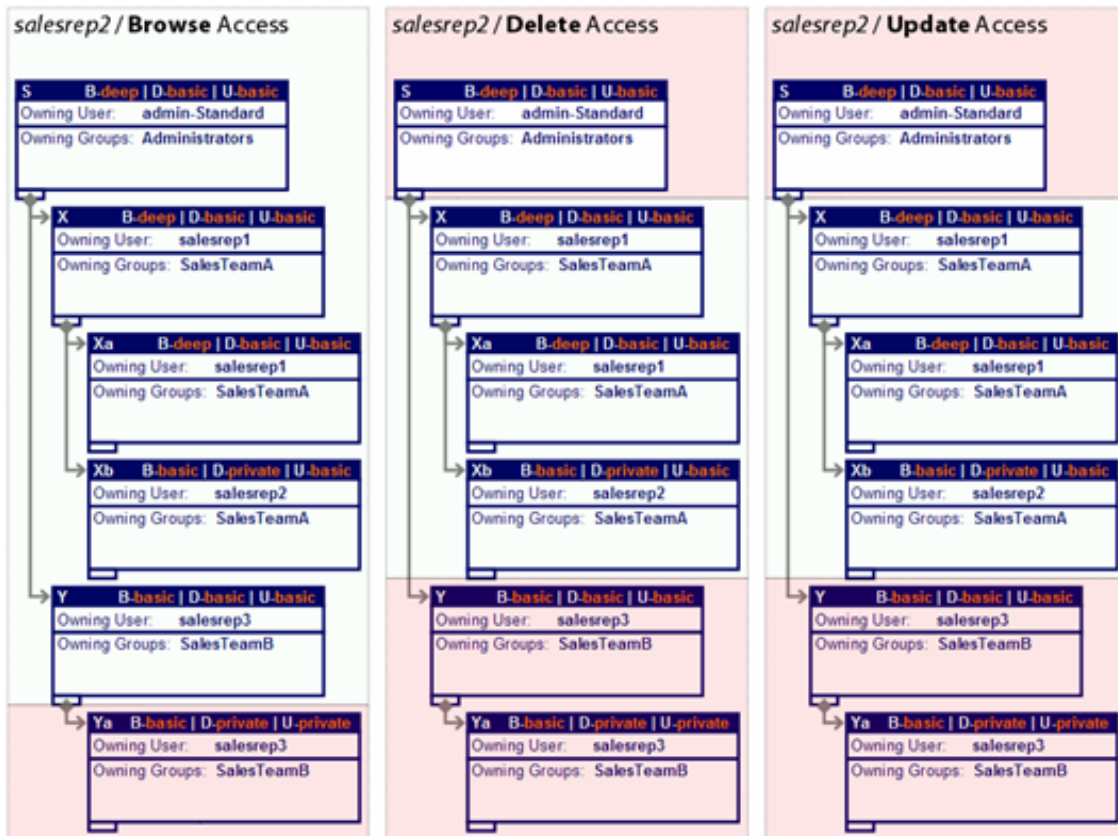
Figure 4-7. Access Permissions of User *salesrep1*



**Warning** Please note that the user *salesrep1* is granted permission to delete object *X*; due to the definition of the delete access level deleting object *X* will also delete all its composite objects (recursively!), i.e. *salesrep1* can actually **delete object *Xb* by deleting object *X***! It is not possible, however, that user *salesrep1* navigates to – for example – object *Xb* and then deletes object *Xb* as delete access is not granted on object *Xb* (even though browsing to object *Xb* is possible).

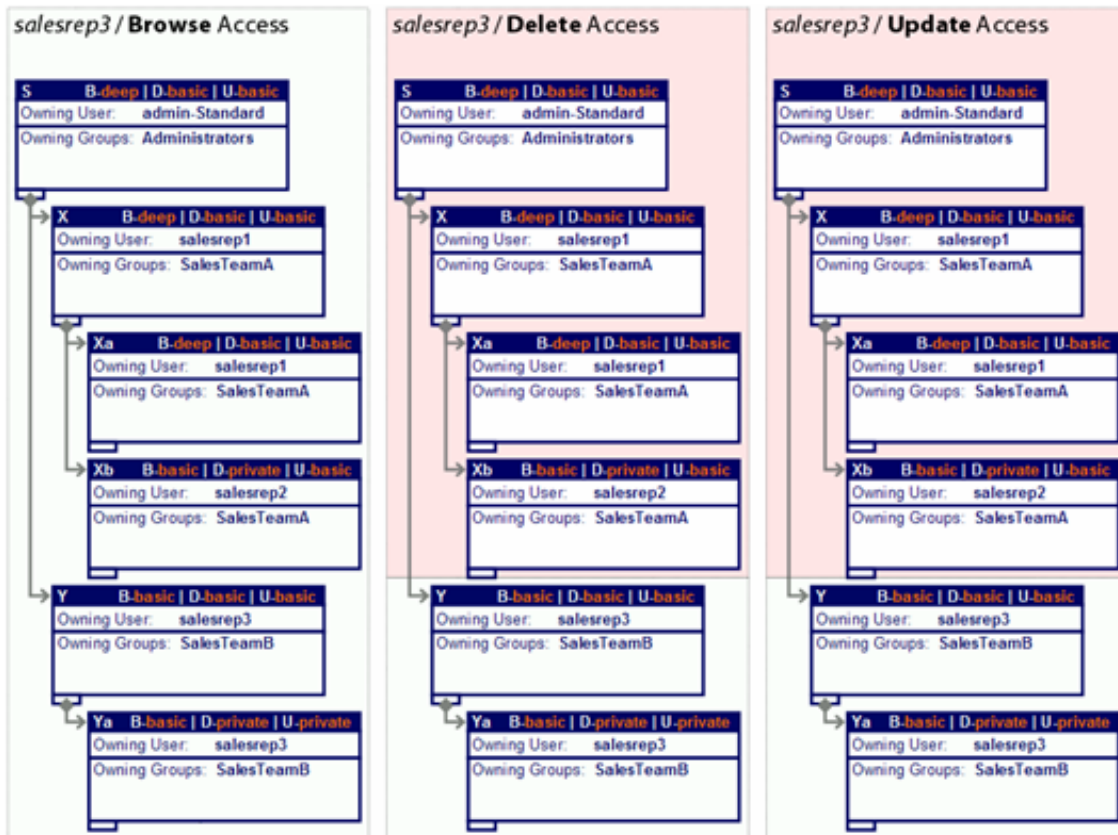
On the following chart, green indicates that access is granted to the user *salesrep2*, red indicates that access is not granted to *salesrep2*:

Figure 4-8. Access Permissions of User *salesrep2*



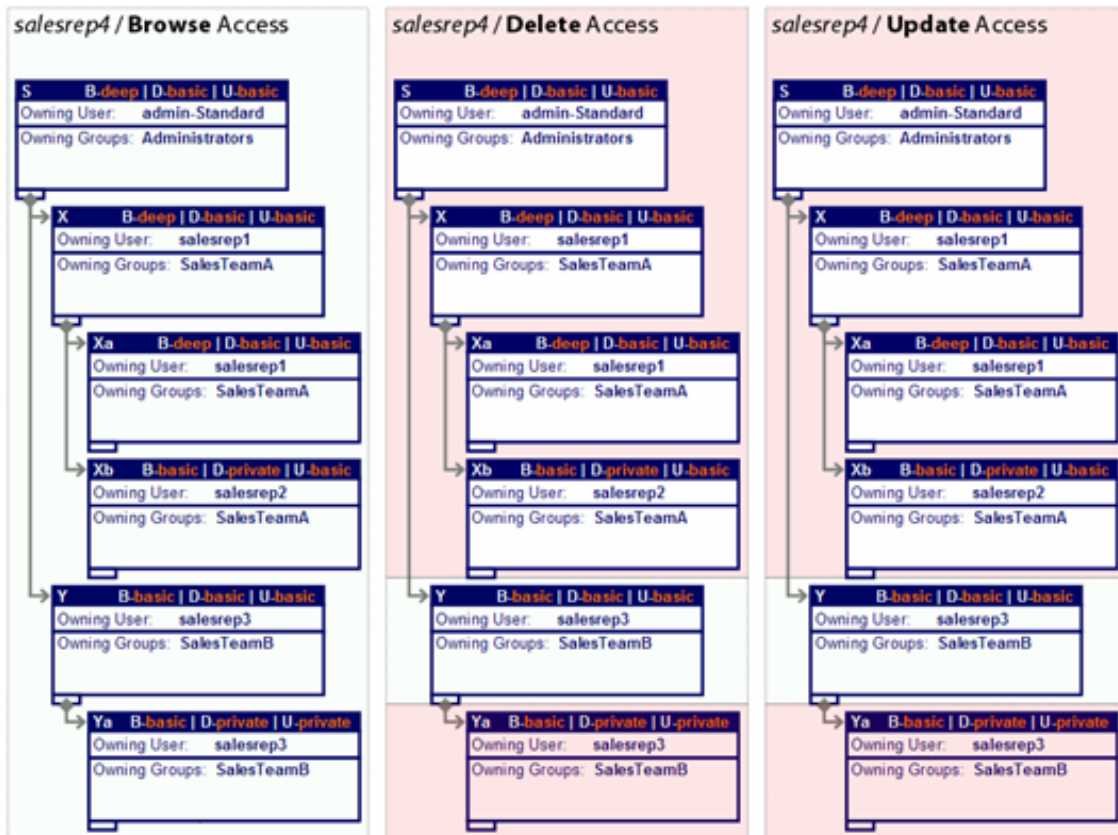
On the following chart, green indicates that access is granted to the user *salesrep2*, red indicates that access is not granted to *salesrep2*:

Figure 4-9. Access Permissions of User *salesrep3*



On the following chart, green indicates that access is granted to the user *salesrep2*, red indicates that access is not granted to *salesrep2*:

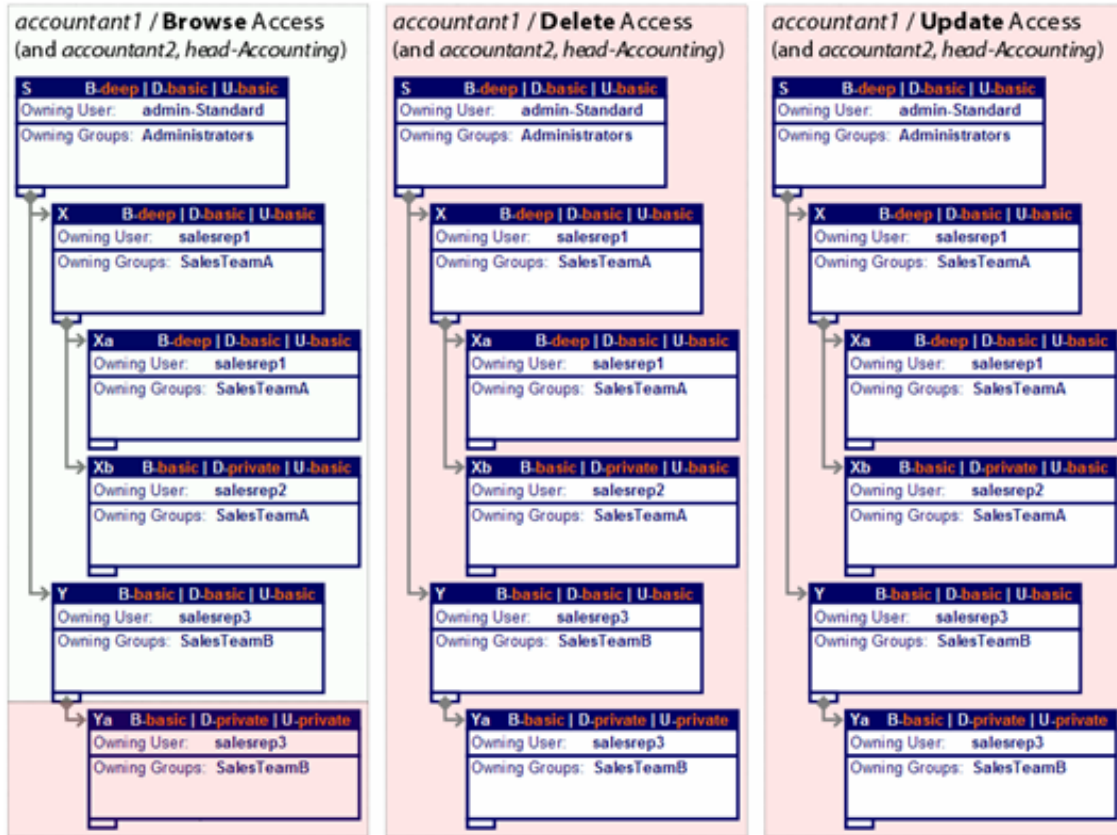
Figure 4-10. Access Permissions of User *salesrep4*



**Warning** Please note that the user *salesrep4* is granted permission to delete object *Y*; due to the definition of the delete access level deleting object *Y* will also delete all its composite objects (recursively!), i.e. *salesrep4* can actually **delete object *Ya* by deleting object *Y***! It is not possible, however, that user *salesrep4* navigates to – for example – object *Ya* and then deletes object *Ya* as delete access is not granted on object *Ya* (even though browsing to object *Ya* is possible).

The following figure shows access permissions of the users *head-Accounting*, *accountant1*, and *accountant2*. Green indicates that access is granted, red indicates that access is not granted:

**Figure 4-11. Access Permissions of Users *head-Accounting*, *accountant1*, and *accountant2***



## Model Permissions

Model permissions are not implemented yet.

## Chapter 5. Next Steps

# Appendix A. Appendix

# Bibliography

[01] *openCRX - the leading open source CRM solution*, [opencrx.org](http://www.opencrx.org).

@ <http://www.opencrx.org> (<http://www.opencrx.org>)

[02] *openMDX - The leading open source MDA platform*, [openmdx.org](http://www.openmdx.org).

@ <http://www.openmdx.org> (<http://www.openmdx.org>)