Dimensional Insight

Audit Logging in Diver Platform 7.0

Audit Logging in Diver Platform 7.0

Revision: Doc-DPWBAL-70-121417-01

December, 2017

Contents

Audit Logging

DiveLine provides the ability to record detailed information about who has accessed what data. This logging feature is available for those customers who require more detail (HIPAA for example) than the default DiveLine logs provide. You set auditing by directory or file, for user or group.

You can define audit logging for cBases and Models in Workbench projects similar to how it is defined for classic Models in Solution 6.4. You define a set of audit-triggering dimensions, plus additional columns to log when an audit occurs. Below are descriptions of how you apply audit rules.

NOTE: Audit controls are established in Workbench by using an Access Control file. Audit logging defined previously using DI-Config still applies to Models referenced outside of Workbench projects (for example in a 6.4 DiveLine Namespace).

See also:

- Access Control Overview on page 9
- Access Control File Samples on page 43
- Access Control File Code Block on page 48
- Access Control File Tags on page 52

Custom Audit Logging

There are two parts to customizing your audit logs:

- Determining when to snapshot the data—defining the audit triggers.
- Specifying precisely what data to capture—listing the audit columns.

The presence of an audit trigger in viewed data causes the logging of both its value(s), and the name(s) and value(s) of any fields tagged as audit columns in that view.

- **Classic Models in Projects**—only Dimensions can be defined as audit triggers, but they include Dynamic Dimensions as well as Core Dimensions. The audit column may be a Summary, Info Field, Dimension, or calculated column.
- **cBases**—an audit trigger has to be a diveable column (basically anything except doubles). The audit column can be any column.

NOTE: Setting audit columns slows the display performance for any dive window containing that column, as every row in the window is recorded in the log. Storage of the log files will also use more disk space. Take care when setting up audits. The impact on performance depends on the degree of logging performed.

Audit Logging Protocol

Logging triggers whenever an audit trigger field is accessed by creation of a Dive window. If any Dimension in that window, or any of the parent Dimensions in the DivePath are marked as an audit trigger, then the new Dive window is logged. Entries are logged indicating the DivePath to this window (all parent Dimension names and values), and the names and values of all audit columns that appear in it.

NOTE: Calculated columns are evaluated as well when considering what to log. If the definition of the calculated column references an audit column, or references another calculated column which references an audit column, then it is also logged.This includes columns based on a Named Group, if the Named Group is defined using an audit column.

Tagging Columns

To enable field-level logging of a cBase or Model, columns are designated as either an audit trigger or an audit column. By default, columns in a cBase or Model are neither. These designations are defined in the access control file.

For traditional Models, you tag either a core Dimension or an Info Field as an audit trigger. Logging of a window is only activated when you dive on the audit trigger. For an Info Field trigger, the Info Field is promoted to a Dynamic Dimension; then, if part of the DivePath, that Dynamic Dimension will trigger an audit. On the other hand, an Info Field that is tagged as an audit trigger but appears as an ordinary Info Field in a Dive window is treated as an audit column, not an audit trigger.

For cBases, any diveable column can be tagged as an audit trigger. The Build script might declare that a particular diveable column should appear in the ProDiver console by default or not. The Build script may declare one column is an Info off another. As with Models, a dive on a cBase must include the audit trigger to cause the window to be logged.

Log File Formats

Logging output is broken up into several text files located in `\dl-dataroot\logs`. Each of these files contains rows of data tied together with a single session-event number. The session number is unique to a single user logging into a DiveLine. The event number is unique to a newly created window (for example, a dive creates a new tabular window). The combination of session and event uniquely associates an entry with a user's window. Using several files at once stores audit data in a denormalized format. It takes up some additional disk space, but makes the log files more readable and easier to parse. The output files are divided by date, with one directory for each month and one file for each day. The columns within each file are tab-delimited and are detailed in the following tables.

Audit Log Files

The contents of the detailed audit logs generated by DiveLine are outlined below.

Each of these files appears in the `dl-dataroot\logs\audit_<component>\YYYYMM` directory. For example:

D:\di\solution\dl-dataroot\logs\audit_session\201704\audit_session-20170409.log
D:\di\solution\dl-dataroot\logs\audit_trigger\201704\audit_trigger-20170409.log

```
D:\di\solution\dl-dataroot\logs\audit_divepath\201704\audit_divepath-20170409.log
D:\di\solution\dl-dataroot\logs\audit_column\201704\audit_column-20170409.log
D:\di\solution\dl-dataroot\logs\audit_divepath\201704\audit_dive_request-20170409.log
```

Audit Session Log

The audit session log contains the top-level information about audit logging. Each row represents a single session-event.

Column	Description
Session-Event	A unique string composed of a user's Session ID concatenated with an Event number, in the format of YYYY-MM-DD_HH:MM:SS_PID_EVENTNUMBER. A Session ID contains a time stamp from the beginning of the user's session plus a process id number. Events are numbered starting at 1. Each time a user creates a window, the Event number for that session goes up by one. For example: 2011-04-28_10:34:48_3908_0001
TimeStamp	A string in the format YYYY-MM-DD HH:MM:SS that identifies the date and time of the logged session. For example: 2011-04-28 10:34:48
User	The userid logged in for the audit-related session. For example: jsmith
DataSource	The source of data being used in the audit-related session. For example: medications.mdl
WindowType	The type of dive done in the audit-related event (i.e., Window, MultitabWindow, MultiCrosstabWindow, CrosstabWindow, ReportPaletteCell). For example: MultitabWindow

Audit Trigger Log

The audit trigger log contains records of activated triggers, each associated with a session-event. See the links above for Access Control File topics for more information on how to define these triggers.

Column	Description
Session-Event	A unique string composed of a user's Session ID concatenated with an Event number, in the format of YYYY-MM-DD_HH:MM:SS_PID_EVENTNUMBER. A Session ID contains a time stamp from the beginning of the user's session plus a process id number. Events are numbered starting at 1. Each time a user creates a window, the Event number for that session goes up by one. For example: 2008-04-28_10:34:48_3908_0001
DimensionName	A string that identifies the audit-related Dimension. For example: Patient ID
User	The userid logged in for the audit-related session. For example: bmarley

Audit DivePath Log

The audit DivePath log contains records of the DivePath for a Session-Event.

Column	Description
Session-Event	A unique string composed of a user's Session ID concatenated with an Event number, in the format of YYYY-MM-DD_HH:MM:SS_PID_EVENTNUMBER. A Session ID contains a time stamp from the beginning of the user's session plus a process id number. Events are numbered starting at 1. Each time a user creates a window, the Event number for that session goes up by one. For example: 2008-04-28_10:34:48_3908_0001
DimensionName	A string that identifies the audit-related Dimension. For example: Patient Name
DimensionValue	A string that identifies the audit-related Dimension Value. For example: Doe, John
DivePathIndex	An integer, starting at "1", that indicates the location of the Dimension in the DivePath. The highest index in a Session-Event indicates the most recent Dimension. In the case of Dimension counts or values in the ProDiver console, this value is -1. Grouped DimensionValues share the same DivePathIndex. For example: 3

Audit Column Log

The audit column log contains information on data that is marked as an Audit Column. See the links above to Access Control File topics for information on how to specify these columns.

Column	Description
Session-Event	A unique string composed of a user's Session ID concatenated with an Event number, in the format of YYYY-MM-DD_HH:MM:SS_PID_ EVENTNUMBER. A Session ID contains a time stamp from the beginning of the user's session plus a process id number. Events are numbered starting at 1. Each time a user creates a window, the Event number for that session goes up by one. For example: 2008-04-28_10:34:48_3908_0001
ColumnNumber	An integer that indicates the audit-related column within the Tabular window created by the user. If audit columns have been specified, any column that contains Dimension values is also logged. This allows for a partial reconstruction of the window the user observed. For example: 3
RowNumber	An integer that indicates the audit-related row within the Tabular window created by the user. For example: 24
ColumnName	Indicates the name of the column that is tagged for logging. The ColumnName is one of the Audit Columns as specified in the Access Control File (or Model's ACL if a 6.x file). For example: Primary HCC
ColumnValue	Indicates the value of the column that is tagged for logging. The ColumnValue is the associated value that the user sees in the window. ColumnValue may denote a Summary or Dimension value. For example: HCC19 - Diabetes

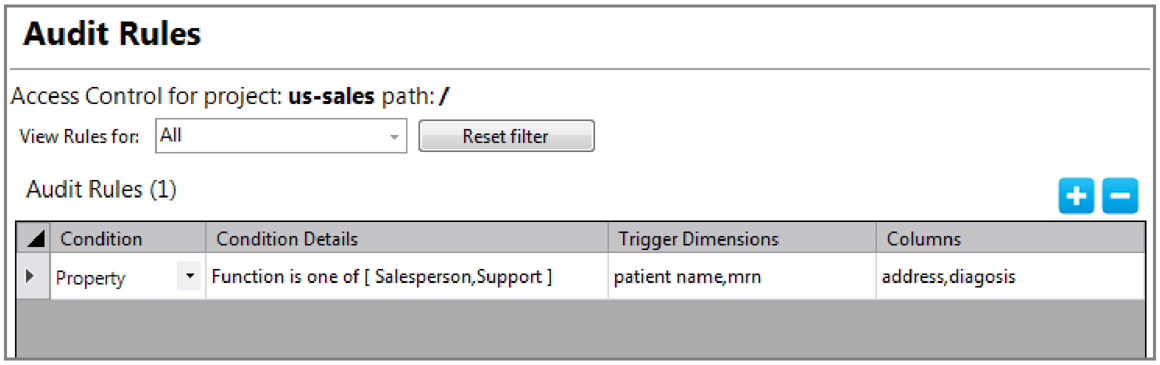
Setting Audit Rules

The **Audit Rules** sub-tab of the access tab allows you to set *Audit* access rules for your Workbench project.

To set audit rules

1. Right-click the project root (or a sub-directory) and click **Edit Access Control** to open the **Access for** / tab.
2. In the **Access for** / tab, click **Audit Rules**.

   The **Audit Rules** tab opens.

   Audit Rules

   Access Control for project: **us-sales** path: **/**

   View Rules for: All

   Audit Rules (1)

Condition	Condition Details	Trigger Dimensions	Columns
Property	Function is one of [Salesperson,Support]	patient name,mrn	address,diagosis

3. Set the **Condition** and **Condition Details** (the *who* the rule applies to).
4. Set audit rules by listing trigger dimensions and the columns viewed to cause a log entry (separated by commas with no spaces).
5. Save the tab.

   NOTE: In order to have the column's accessed values appear in the audit log, you must list the columns in both the **Trigger Dimensions** and **Columns** sections for each audit rule.

See also:

- Audit Logging on page 4
- Configuring Access Control on page 26
- Access Control Overview below
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

Access Control Overview

There are several aspects to setting access control (that is, configuring the security) in Workbench projects. This topic gives a general work flow and links to specific topics for steps.

At the basic level, you grant or restrict project access to users and groups.

At a more advanced level, you use Workbench properties when configuring **Access Control** to grant or restrict access to specific files, folders, or columns and rows within a cBase or classic Model, based on the property values assigned to users and groups.

The general work flow for setting access control is:

1. Create users.
2. Create groups and assign users to them.
3. Create properties.
4. Set project access for users and groups.
5. Assign properties and specific values to users and groups.
6. Use these properties to configure access control for specific folders in the Workbench Explorer (see Configuring Access Control on page 26). You can also use these property values in other scripts.

NOTE:

- DI recommends using properties as the main way to set access control rules as it allows you to separate the rules from data values (see Properties: Separating Access Control Rules from Data Values on page 21).
- When using classic Diver Solution Models and Workbench projects, access control is as described above. The access control tab includes a **Model Access** tab for these settings (see Setting Model Access on page 32).
- When using classic Models with a virtual project in Workbench via the 6.4 DiveLine Namespace or 6.4 Production Sandbox, existing DiveLine **Access Control Lists**[1] ACLs created using DI-Config will be applied by the 7.0 clients. This includes the scenario where a project has an alias to the 6.4 DiveLine Namespace or 6.4 Production Sandbox project. To edit these ACLs, use version 7.0 DI-Config.

See also:

- About Access Control on the next page
- Access Control Interface on page 13
- Access Control Process on page 14
- Access Control Categories on page 16
- Directory Overview Tab on page 18
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Configuring Access Control on page 26

[1]access control list. The security tool that DiveLine 6.x uses to control user and group access to the server. DiveLine 7.0 applies access control rules to projects, and can apply ACLs to non-project resources that use the 6.4 DiveLine namespace.

- Access Control File Tags on page 52
- Access Control File Code Block on page 48

About Access Control

Access control consists of rules you set to control the security of a Workbench project. These rules can be set at the project level or directory (folder) level, and includes the ability to set who can access the parts of your data sets contained in cBases or models. In addition, audit rules can capture (in a log) which users have seen particularly sensitive data.

Workbench projects default to allow full access to administrative users and completely restrict access to non-administrative users. That is, access control rules default to the most restrictive for non-administrative users. You can grant the non-administrative users permissions to access directories and portions of your data sets with the access control rules.

These rules fall into the following categories:

- File Access
- cBase Access
- Model Access
- DiveTab Access
- Audit Rules

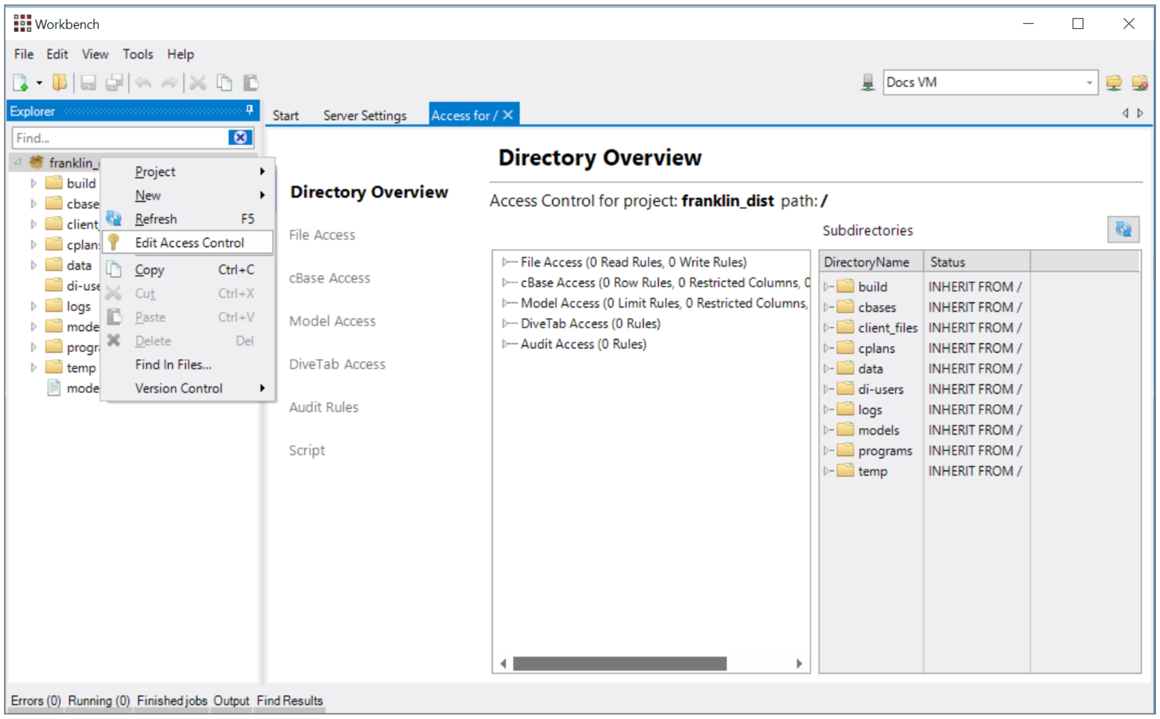
Each rule is set either without a condition (applies to all users) or with a condition set by user, group, or property. The condition types are:

- All Users—Applies to all users (default)
- Group—Applies to everyone within the named group
- User—Applies to the specific user
- Property—Applies to any user or group that has the specified property and value(s) pair

Opening the Access Control Tab

1. Right-click the project or folder in Workbench Explorer and click **Edit Access Control**.

   The **Access for /** tab opens. Note that the forward slash ("/") indicates this is the project root and a directory name appears when opening from a folder (for example, **Access for /cbases**).
2. Click the appropriate sub-tab to set the access control rules you need (as shown in this sample).



Directory Overview

Access Control for project: **franklin_dist** path: **/**

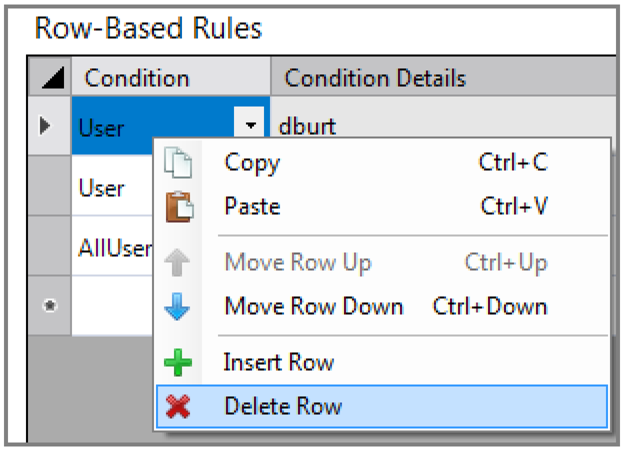
- File Access (0 Read Rules, 0 Write Rules)
- cBase Access (0 Row Rules, 0 Restricted Columns, 0
- Model Access (0 Limit Rules, 0 Restricted Columns,
- DiveTab Access (0 Rules)
- Audit Access (0 Rules)

Subdirectories

DirectoryName	Status
build	INHERIT FROM /
cbases	INHERIT FROM /
client_files	INHERIT FROM /
cplans	INHERIT FROM /
data	INHERIT FROM /
di-users	INHERIT FROM /
logs	INHERIT FROM /
models	INHERIT FROM /
programs	INHERIT FROM /
temp	INHERIT FROM /

NOTE:

- After rules are set in the rules tables, there are context-menu commands for editing the table rows.



Row-Based Rules

Condition	Condition Details
User	dburt
User	
AllUser	

- Each new rule is set on a new row within the specific access sub-tab.

Note on inheritance:

Access control is either set on a directory or project, or it is not. This is done with all access control categories. If access control is not defined for a directory, then access control rules are inherited from the nearest ancestor that has any rules defined. It is not possible to inherit access for only one category. For instance, it is not possible to inherit file access control without also inheriting cBase access control. If no ancestor has access control defined, then the default (restrictive) access is used.

When access control is set on a project or directory, the underlying access script is saved in the project encoded in UTF-8.

See also:

- Access Control Interface below
- Access Control Process on the next page
- Access Control Categories on page 16
- Directory Overview Tab on page 18
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Configuring Access Control on page 26

NOTE: **Access Control Lists**[1] (ACLs) for 6.4 DiveLine Namespace and 6.4 Production Sandbox projects are maintained using the version 7.0 **DI-Config**[2].

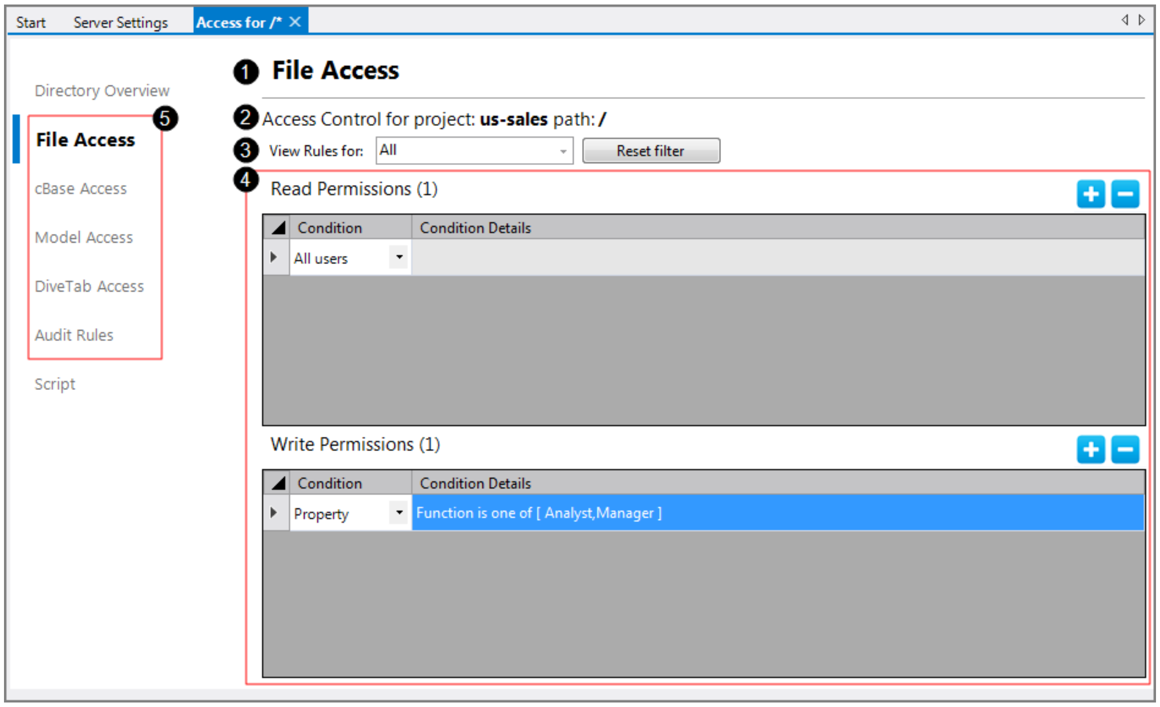
Access Control Interface

Access control rules are set within the access tab (right-click a project or directory and click **Edit Access Control**). This interface consists of several sub-tabs where you set the rules per category. In the sample screen shown here, note the following (numbers in the text refer to numbers in the sample screen):

- **Category name** (1)—Displays the name of the selected sub-tab
- **Project path** (2)—Displays the project path (as shown in the graphic as **us-sales** path: / where the forward slash ("/") represents the root of the project **us-sales**)
- **Filter drop-down list** (3)—Displays a listing of users and groups that you can use to filter the listings when you have numerous rules
- **Rules tables** (4)—Displays one or more tables, based on the selected sub-tab where you set the individual rules
- **Sub-tabs** (5)—Selects and displays the selected access control category

[1]access control list. The security tool that DiveLine 6.x uses to control user and group access to the server. DiveLine 7.0 applies access control rules to projects, and can apply ACLs to non-project resources that use the 6.4 DiveLine namespace.

[2]The DiveLine subcomponent that allows an administrator to configure DiveLine options using a Windows user interface. In 7.0, DI-Config functionality is part of the Workbench server settings.


Access Control for project: **us-sales** path: **/**

View Rules for: All

Read Permissions (1)

Condition	Condition Details
All users	

Write Permissions (1)

Condition	Condition Details
Property	Function is one of [Analyst,Manager]

See also:

- About Access Control on page 11
- Access Control Process below
- Access Control Categories on page 16
- Directory Overview Tab on page 18
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Configuring Access Control on page 26

Access Control Process

Setting access control to ensure your files and data are secure has many parts. The rule settings in the access tab is one of the final steps. Review the overall process to understand prerequisite pieces you may need to prepare (see Access Control Overview on page 9) .

Work flow for setting access control rules:

- At the project level, set file access to allow *read* and *write* (if applicable) permissions for non-administrative users.

  This allows non-administrative users to see into the project. It allows them to see all of the project directories and you will need to restrict them from any directories you do not want them to see.

- Restrict non-administrative users from any directories you do not want them to see.

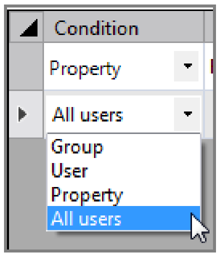
  This is achieved by removing the **Inherit from ancestor** check mark from the **Directory Overview** tab of the access tab for each folder you do not want them to see.

  Basically, you first grant full *read* and *write* permissions for file access to the entire project, then restrict access to any directories you do not want the user to see.
- Set the data-specific access rules to control what each user and group can see within the data set.

  For data sets (cBases and models), you first restrict columns and then allow specific users or groups access (most often through property/values assignments). You can also set limit rules for rows. For DiveTab access rules, you first restrict DiveTab areas (buttons) and then allow specific users or groups access to them. When DiveTab applications access cBases, the row and columns rules apply to the display of DiveTab data pages.
- Set audit rules if required.

NOTE:

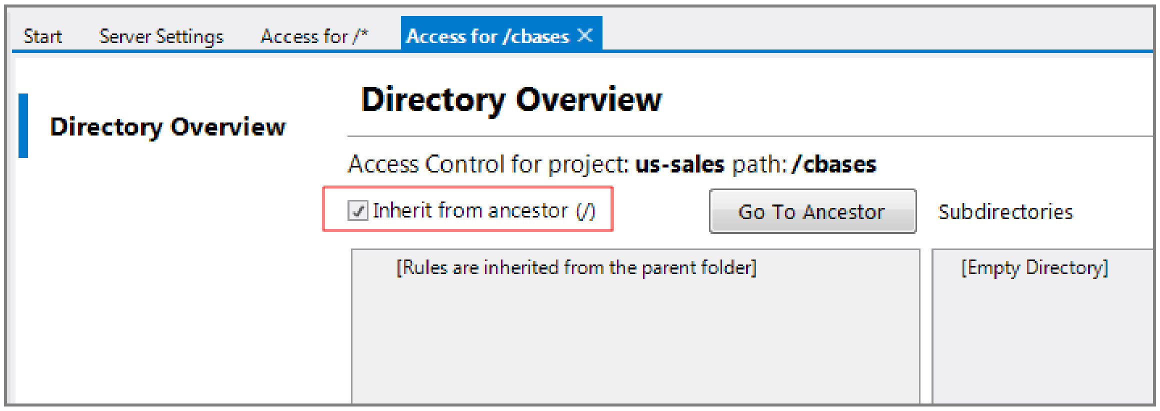
- Rules which include a conditional tag (*if-property*, *if-group*, *if-user*) are considered *conditional rules* (see Access Control Conditions on page 19).
- These conditional *if* tags are only visible as such in the script. In the interface, they are presented as a choice in the **Condition** column of the rules tables as shown here.

  NOTE: The *All users* choice sets that rule without a condition so that the rule applies to all users. This is the default setting that places no *if* statement for the rule in the underlying script (see Underlying Access Control Script on page 40)
- When looking at the access file, before making access decisions, DiveLine will scan all the conditional rules and test each to see if it applies to the current user attempting to access data. If the rule does not pass the condition, the rule is removed from consideration—it would be as if the rule were not present at all.
- Only the rules with satisfied conditions or without any condition at all are processed to determine effective access.

Notes on inheritance:

- Each directory (folder) below the project level defaults to have the **Inherit from ancestor** check mark turned on and does not display any of the category sub-tabs as shown here.



Directory Overview

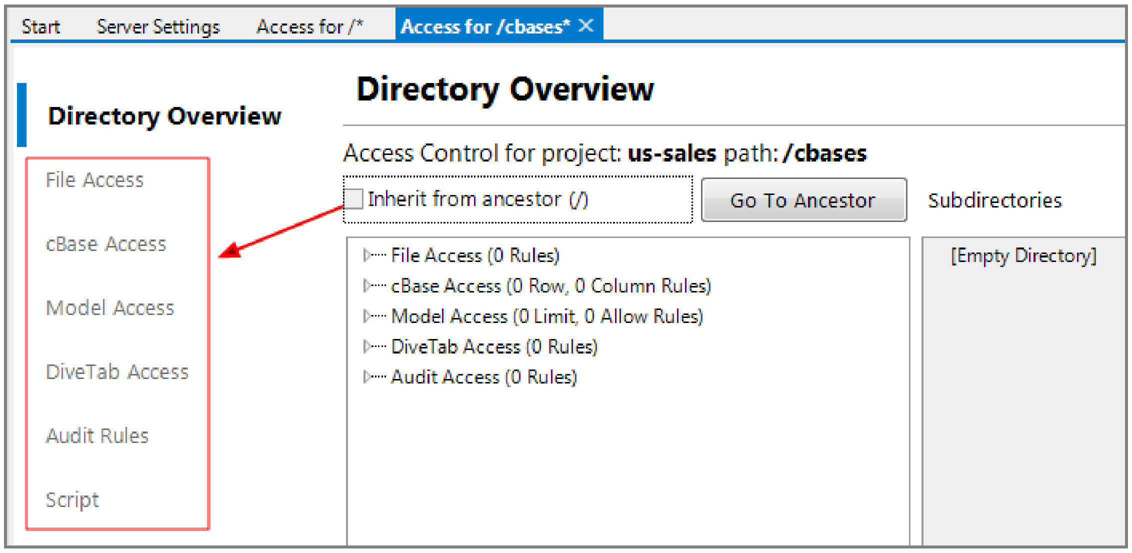
Access Control for project: **us-sales** path: **/cbases**

☑ Inherit from ancestor (/)

[Rules are inherited from the parent folder]

[Empty Directory]

- To make any changes to the access control rules for this directory, you must first deselect the **Inherit from ancestor** check mark, which then exposes all of the sub-tab categories as shown here.



Directory Overview

Access Control for project: **us-sales** path: **/cbases**

☐ Inherit from ancestor (/)

File Access (0 Rules)
cBase Access (0 Row, 0 Column Rules)
Model Access (0 Limit, 0 Allow Rules)
DiveTab Access (0 Rules)
Audit Access (0 Rules)

[Empty Directory]

See also:

- About Access Control on page 11
- Access Control Interface on page 13
- Access Control Categories below
- Directory Overview Tab on page 18
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Configuring Access Control on page 26

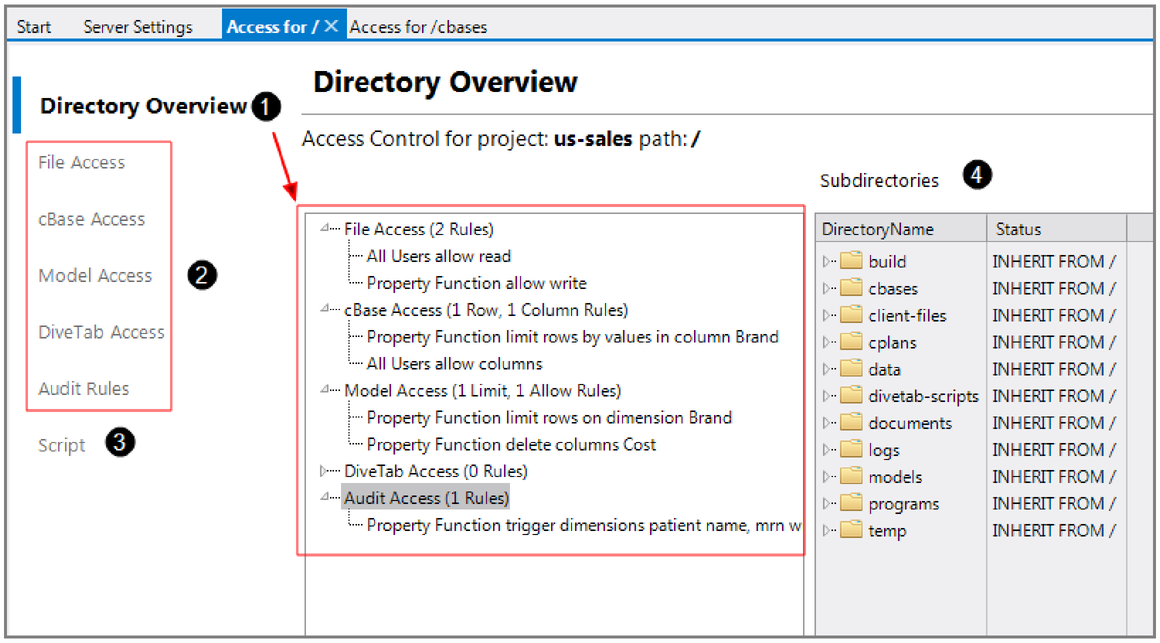
Access Control Categories

Access control rules can be set for the following categories:

- File Access
- cBase Access
- Model Access
- DiveTab Access
- Audit Rules

The access control interface has a separate sub-tab for each of the categories where you set the rules.

A sample screen is shown here. You can view a summary of existing rules (1), set access rules per category (2), work with the underlying script (3), and inspect the status of any existing subdirectories (4). Numbers here refer to the numbers in the example screen.



Directory Overview

Access Control for project: **us-sales** path: **/**

- File Access (2 Rules)
  - All Users allow read
  - Property Function allow write
- cBase Access (1 Row, 1 Column Rules)
  - Property Function limit rows by values in column Brand
  - All Users allow columns
- Model Access (1 Limit, 1 Allow Rules)
  - Property Function limit rows on dimension Brand
  - Property Function delete columns Cost
- DiveTab Access (0 Rules)
- Audit Access (1 Rules)
  - Property Function trigger dimensions patient name, mrn w

DirectoryName	Status
build	INHERIT FROM /
cbases	INHERIT FROM /
client-files	INHERIT FROM /
cplans	INHERIT FROM /
data	INHERIT FROM /
divetab-scripts	INHERIT FROM /
documents	INHERIT FROM /
logs	INHERIT FROM /
models	INHERIT FROM /
programs	INHERIT FROM /
temp	INHERIT FROM /

NOTE: The tab label displays the path to where this access control tab resides. In the example, it is at the project root that is represented by the forward slash (/) with **Access for** /.

DI recommends you set your access control rules at the root of the project, where the tab name is **Access for** / (as in the example). All the other directories (and any subdirectories) default to inherit access control rules from the ancestor, which is most often the project root. You can set access control rules on individual directories, by deselecting the **Inherit from ancestor** check mark for the directory's access control tab.

Each rule category may have multiple access control rule types. For example, for the file access category, there are *read* and *write* rules and for the cBase access category, there are *row-based* and *column-based* rules.

All access control rules have two parts:

- Condition—Who does this rule apply to? See Access Control Conditions on the next page.
- Effect—What does this rule do? See Access Control Effects on page 22.

The effect can be complex, since it is a combination of the rule type and the parameters of the rule.

Effects vary depending on category and the rule types selected. For ease of maintenance, consistency of usage is encouraged within a project.

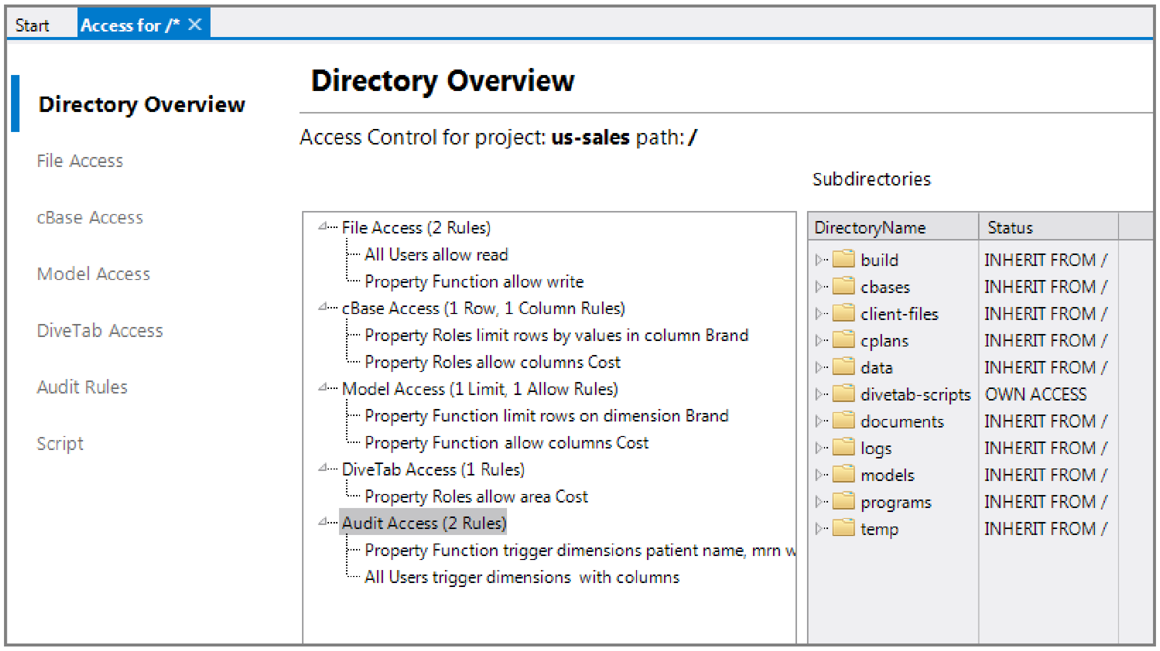
See also:

- About Access Control on page 11
- Access Control Interface on page 13
- Access Control Process on page 14
- Directory Overview Tab below
- Access Control Conditions on the next page
- Access Control Effects on page 22
- Configuring Access Control on page 26

Directory Overview Tab

The **Directory Overview** tab of an access tab provides a summary of any existing access control rules.

The screen here shows a sample where there are rules set for each category.



Directory Overview

Access Control for project: **us-sales** path: **/**

Subdirectories

File Access (2 Rules)
- All Users allow read
- Property Function allow write

cBase Access (1 Row, 1 Column Rules)
- Property Roles limit rows by values in column Brand
- Property Roles allow columns Cost

Model Access (1 Limit, 1 Allow Rules)
- Property Function limit rows on dimension Brand
- Property Function allow columns Cost

DiveTab Access (1 Rules)
- Property Roles allow area Cost

Audit Access (2 Rules)
- Property Function trigger dimensions patient name, mrn w
- All Users trigger dimensions with columns

DirectoryName	Status
build	INHERIT FROM /
cbases	INHERIT FROM /
client-files	INHERIT FROM /
cplans	INHERIT FROM /
data	INHERIT FROM /
divetab-scripts	OWN ACCESS
documents	INHERIT FROM /
logs	INHERIT FROM /
models	INHERIT FROM /
programs	INHERIT FROM /
temp	INHERIT FROM /

NOTE:

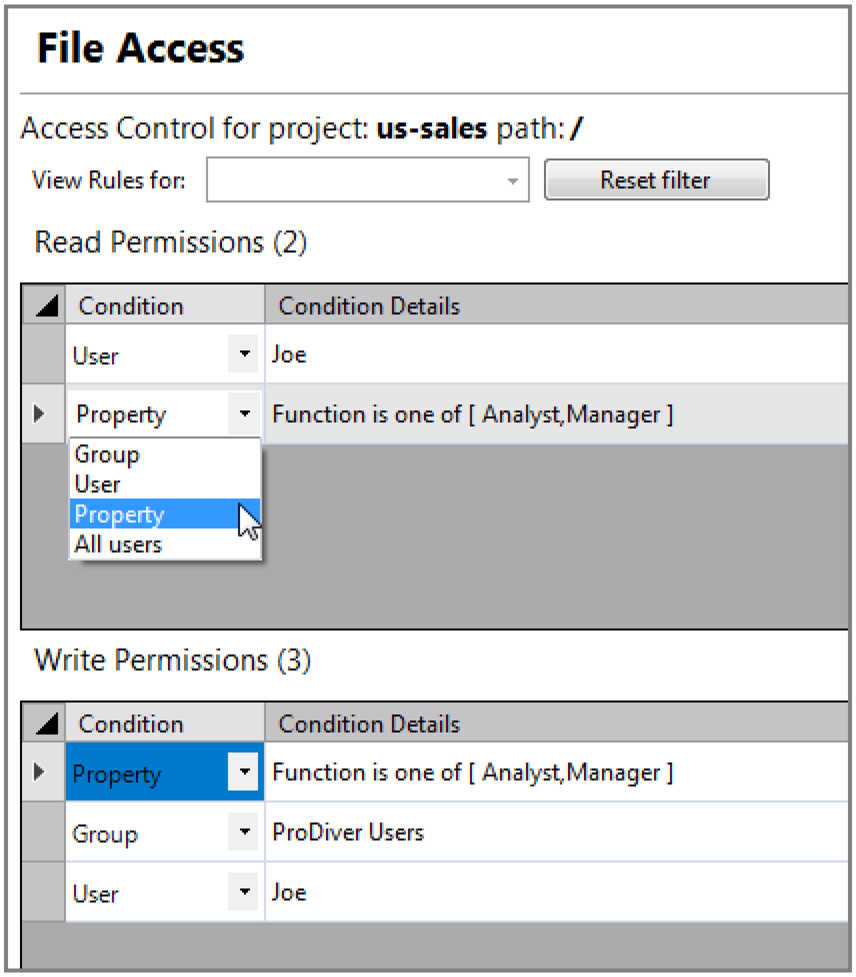
- You can expand a category to see more details. Full details are available from the individual category tabs.
- The **Subdirectories** section on the right shows the access control status of each directory below the currently selected directory (the example shows the project root selected). A directory will either inherit from the ancestor (INHERIT FROM/) or have its own rules set (OWN ACCESS). You can right-click a directory here and choose **Edit Access Control** to go directly to the access tab for that directory.
- You can click refresh for the **Subdirectories** section to ensure any added directories and **Status** changes appear.

See also:

- About Access Control on page 11
- Access Control Interface on page 13
- Access Control Process on page 14
- Access Control Categories on page 16
- Directory Overview Tab on the previous page
- Access Control Conditions below
- Access Control Effects on page 22
- Configuring Access Control on page 26

Access Control Conditions

The access control **Condition** and **Condition Details** fields describe the *to whom* the rule applies. These condition choices are the same for every rule category. An example for a file access rule set is shown here.


File Access

Access Control for project: **us-sales** path: **/**

View Rules for:

Read Permissions (2)

Condition	Condition Details
User	Joe
Property	Function is one of [Analyst,Manager]

Write Permissions (3)

Condition	Condition Details
Property	Function is one of [Analyst,Manager]
Group	ProDiver Users
User	Joe

NOTE: To set rules for multiple conditions, you add a new row to the rules table for each condition.

The following table details the four different condition types.

Condition	Condition Details	Description
Group	Provides the list of groups on the DiveLine	Rule applies to everyone in the selected group
User	Provides the list of users on the DiveLine	Rule applies to the selected user
Property	Provides the list of properties (and values) on the DiveLine	Rule applies to any user or group that has a given property and value (s) pair
All Users	n/a	Rule applies to everyone

Using the *Property* condition allows you to separate the access rules from the users (see Properties: Separating Access Control Rules from Data Values below).

See also:

- About Access Control on page 11
- Access Control Interface on page 13
- Access Control Process on page 14
- Access Control Categories on page 16
- Directory Overview Tab on page 18
- Access Control Conditions on page 19
- Access Control Effects on the next page

Properties: Separating Access Control Rules from Data Values

Although the access control rules allow you to explicitly include dimension values using the `limit-dimension` and `limit-rows-by-values` tags, DI recommends that you do so only on a limited basis.

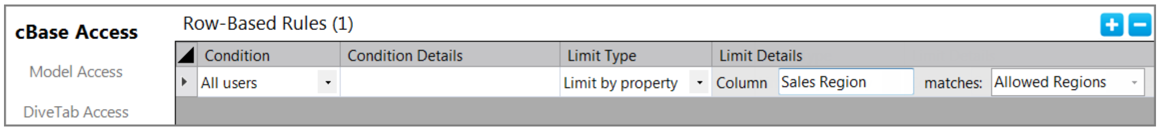
As a best practice, DI recommends that you separate the access control rules from any data values. You can achieve this separation by using **properties**. By separating the access control rules from the data values, you can set the rules once (based on the properties) and not touch them again. Once the access rules are set, security is addressed at the user and group level by property values assignments. As users come and go, or change roles in the organization, you adjust the property value assignments (either for the specific user or for the groups the user is a member of). Not needing to modify access *rules*, but just the *assigned values*, minimizes errors that could upset existing user's security settings.

You define properties on the server and associate values (**Server Setting** > **Properties**). You define rules using those properties. When you create users or groups, all properties are available. If you want a property to apply to a user or group, you set the assigned values as part of the profile (**Server Settings** > **Users** > <**user**> > **Properties** tab; **Server Settings** > **Groups** > <**group**> > **Projects and Properties** tab). When users log in, access rules are applied based on their assigned properties and values. That is, when the system determines a particular user's property values assignments, it takes the *union* of all the values contributed by assignments for the groups this user is a member of, and any explicit assignment for the particular user.

NOTE: It is always best to use role-based assignment when possible. If you tie rules to a particular user, you create a maintenance headache as people come and go. If all access is done using groups, it is simple to move users in and out of groups as needed. This means doing property value assignments at the group level whenever possible.

For example:

Suppose you have a cBase with *Sales Region* data. The *Salespersons* are allowed to see all data for their region. You could define a property called *Allowed Regions* on the server (values are the individual regions), and create the rule to limit access based on that property. For example, define a row based cBase rule for all users that limits the column *Sales Region* to values in the *Allowed Regions* property.



Row-Based Rules (1)

Condition	Condition Details	Limit Type	Limit Details
All users		Limit by property	Column: Sales Region matches: Allowed Regions

The *Salespersons* are assigned to groups, and each group has the *Allowed Regions* property set to the value of the region name; for example the group *North* has the value *North*, and the group *Boston* has the value *Boston*. After configuration in the GUI, the script includes the following:

```
cbase-access {
  limit-rows-by-property {
    column "Sales Region"
    property "Allowed Regions"
  }
}
```

Each *Salesperson* that opens the cBase sees data according to the assigned values for the *Allowed Regions* property.

TIP: Property value assignments can be set from an external file. You could for example, have a nightly external process that creates a flat file used to update property values and assignments. Once set, the rules do not change—just the assignment of values to properties.

See also:

- Configuring Access Control on page 26

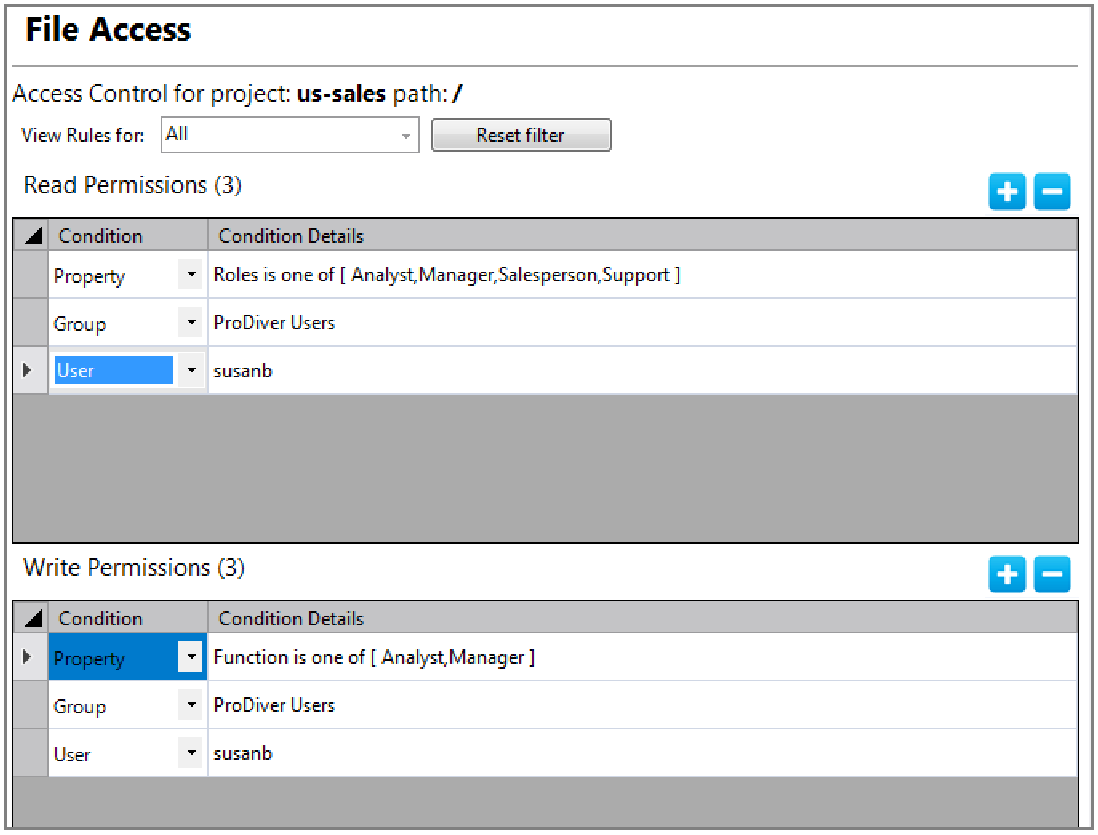
Access Control Effects

Each access rule category uses the same possible *conditions* and *condition details* (the *to whom* the rule applies—see Access Control Conditions on page 19). However, the available *effects* of each rule differs per category. Below are descriptions of the *effect* settings available for each of the access categories.

File Access

Consists of two setting types: *read permissions* and *write permissions*. This allows you to set who can *read* and *write* to a project or directory. See Configuring Access Control on page 26 and Setting File Access on page 27.

Example



Access Control for project: **us-sales** path: **/**

View Rules for: All

Read Permissions (3)

Condition	Condition Details
Property	Roles is one of [Analyst,Manager,Salesperson,Support]
Group	ProDiver Users
User	susanb

Write Permissions (3)

Condition	Condition Details
Property	Function is one of [Analyst,Manager]
Group	ProDiver Users
User	susanb

NOTE: Setting *read* and *write* permissions at the project level gives users access to all directories in the project. This requires that you remove *read* access to any directories in the project you do not want a user to see. This limits what appears in the ProDiver **Open** dialog box.

cBase Access

Consists of two setting types: *row-based rules* and *column-based rules*. See Configuring Access Control on page 26 and Setting cBase Access on page 30.

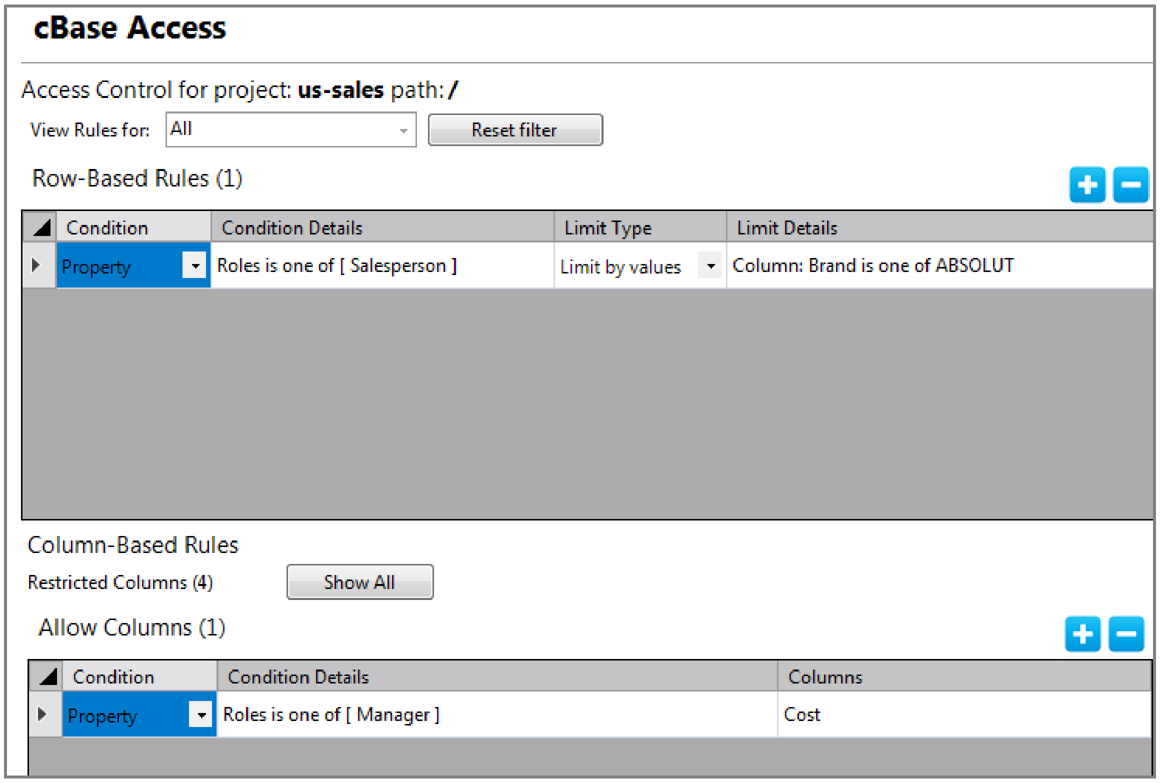
In the *row-based rules* you can set *Limit Type*. Possible settings are:

- Allow All
- Limit By Values
- Limit By Property
- Limit By Username Column
- Limit By Filter

In the *column-based rules*, you first restrict columns and then set rules that allow some users to see these restricted columns.

NOTE: You can use only one **restrict-columns** rule within a **cbase-access** rule set but this can have multiple columns listed. Multiple **allow-columns** rules are allowed so that you can allow back in columns for specific users, groups, or those with a certain property and value(s) applied.

Example



cBase Access

Access Control for project: **us-sales** path: **/**

View Rules for: All | Reset filter

Row-Based Rules (1)

Condition	Condition Details	Limit Type	Limit Details
Property	Roles is one of [Salesperson]	Limit by values	Column: Brand is one of ABSOLUT

Column-Based Rules

Restricted Columns (4) | Show All

Allow Columns (1)

Condition	Condition Details	Columns
Property	Roles is one of [Manager]	Cost

Model Access

Consists of two setting types: *row-based rules* and *column-base rules*. See Configuring Access Control on page 26 and Setting Model Access on page 32.

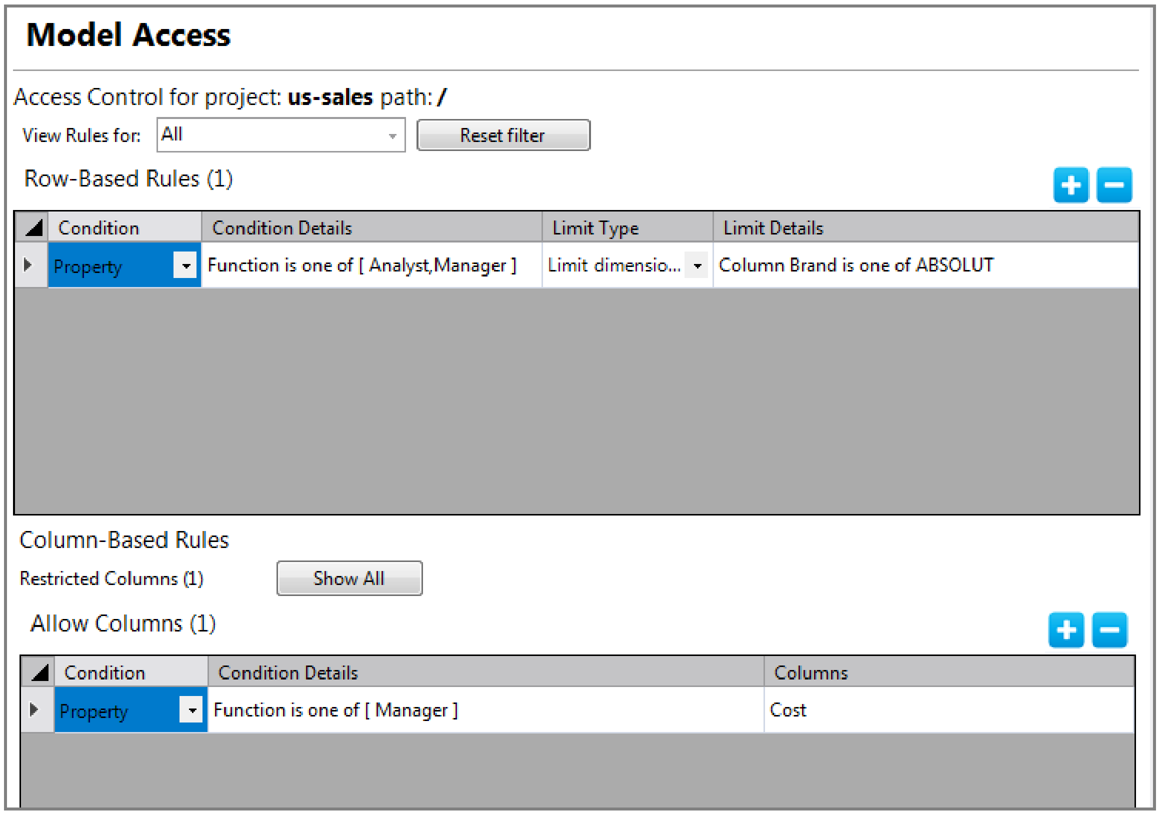
In the *row-based rules*, you can set the *Limit Type*. Possible settings are:

- Limit Dimension By Values
- Limit Dimensions By Groups Column
- Limit Dimensions By Property Value
- Limit Dimension By User Column

In the *column-based rules*, you first restrict columns and then set rules that allow some users to see these restricted columns.

NOTE: You can use only one **restrict-columns** rule within a **model-access** rule set but this can have multiple columns listed. Multiple **allow-columns** rules are allowed so you can allow back in columns for specific users, groups, or those with a certain property and value(s) applied.

Example


Model Access

Access Control for project: **us-sales** path: **/**

View Rules for: All

Row-Based Rules (1)

Condition	Condition Details	Limit Type	Limit Details
Property	Function is one of [Analyst,Manager]	Limit dimensio...	Column Brand is one of ABSOLUT

Column-Based Rules

Restricted Columns (1)

Allow Columns (1)

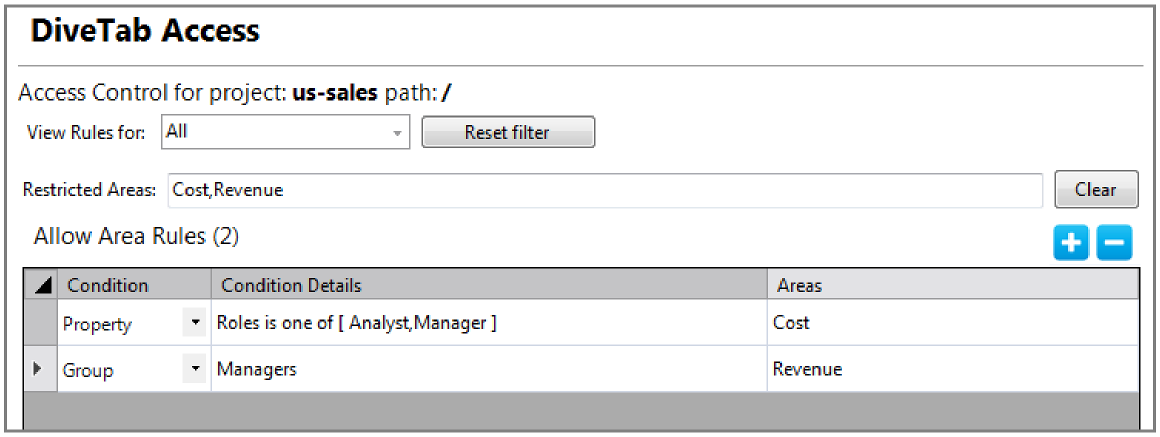
Condition	Condition Details	Columns
Property	Function is one of [Manager]	Cost

DiveTab Access

Consists of one setting to allow access to *areas* (which correspond to buttons in the client). See Configuring Access Control on the next page and Setting DiveTab Access on page 35.

In the allow area rule, you first set areas as restricted and then set rules to allow users access to the area.

Example


DiveTab Access

Access Control for project: **us-sales** path: **/**

View Rules for: All

Restricted Areas: Cost,Revenue

Allow Area Rules (2)

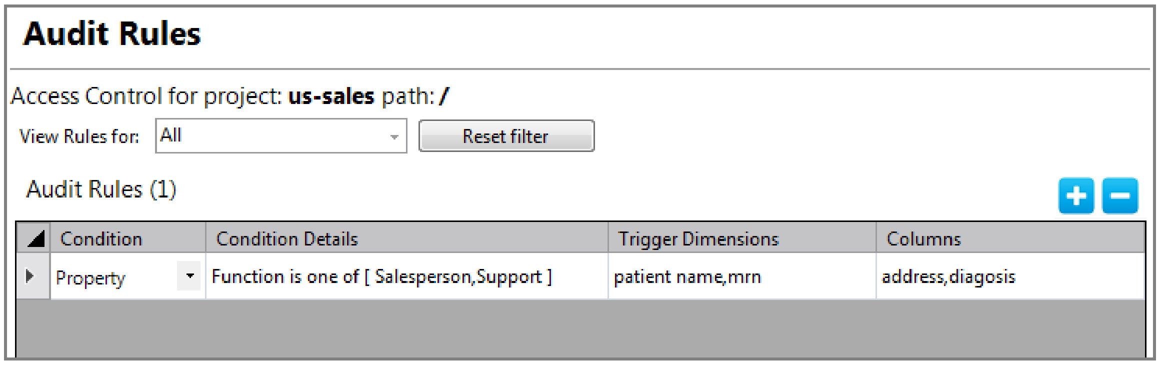
Condition	Condition Details	Areas
Property	Roles is one of [Analyst,Manager]	Cost
Group	Managers	Revenue

Audit Rules

Consists of one setting to set *trigger dimensions* and *columns* to monitor. See Configuring

Access Control on the next page, Setting Audit Rules on page 36, and Audit Logging on page 4.

Example



Audit Rules

Access Control for project: **us-sales** path: **/**

View Rules for: All

Reset filter

Audit Rules (1)

Condition	Condition Details	Trigger Dimensions	Columns
Property	Function is one of [Salesperson,Support]	patient name,mrn	address,diagosis

See also:

- About Access Control on page 11
- Access Control Interface on page 13
- Access Control Process on page 14
- Access Control Categories on page 16
- Directory Overview Tab on page 18
- Access Control Conditions on page 19
- Configuring Access Control below

Configuring Access Control

Setting up access control (security) for your Workbench projects can take many possible forms, depending on the structure of your data and how you want your users to interact with it. See Access Control Overview on page 9 for information about prerequisite items you may need to prepare before working with the access tabs (such as setting up properties, users and groups, as well as project access).

The following are the overall steps for working with the access control sub-tabs. Not all sub-tabs are always applicable in all projects—use only those pertinent to your situation.

1. On the **File Access** tab, set *read* and *write* permissions. See Setting File Access on the next page.
2. On the **cBase Access** tab, set *limit by* details in the **Row-Based Rules** table to limit the row data users can see, and set *restrict/allow columns* rules in the **Column-Based Rules** table to restrict which columns the users can see. See Setting cBase Access on page 30 and Using the Restrict Column Editor on page 33.

3. On the **Model Access** tab, set *limit dimension by* details in the **Row-Based Rules** table to limit row data users can see, and set *restrict/allow columns* rules in the **Column-Based Rules** table to restrict which columns the users can see. See Setting Model Access on page 32 and Using the Restrict Column Editor on page 33.
4. On the **DiveTab Access** tab, set restrictions on what DiveTab *areas* (buttons) the users see. Note that cBase access rules also apply to DiveTab data pages. See Setting DiveTab Access on page 35.
5. On the **Audit Rules** tab, set *trigger dimensions* and which columns cause an entry into the audit logs when these specific columns are viewed. See Setting Audit Rules on page 36.

NOTE:

- If you are using the home project feature, you are required to set a home directory access pattern that sets appropriate access control rules for new users. You can also retrofit existing users for this. The home project likely uses aliases to other projects and you need to ensure these users have project access and that the access control rules are set so users have appropriate access to data.
- If you are not using the home project feature, but are using *user home directories*, you set access rules for those directories so that only the specific user has the ability to *read* and *write* to those directories. You may want to allow other users to have *read* access to these directories but not *write* access.

See also:

- Access Control Overview on page 9
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

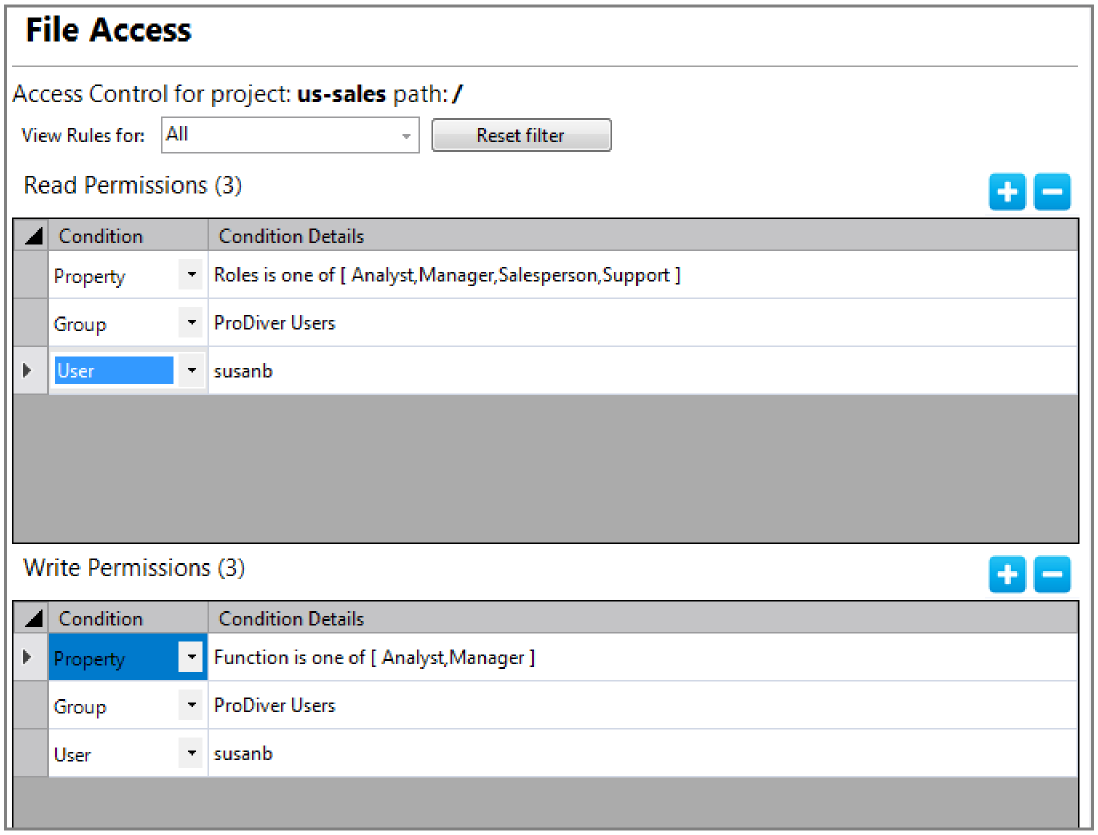
Setting File Access

The **File Access** sub-tab of the access tab allows you to set *read* and *write* access rules for your Workbench project.

To set file access:

1. Right-click the project root (or a sub-directory) and click **Edit Access Control** to open the **Access for/** tab.
2. In the **Access for /** tab, click **File Access**.

   The **File Access** tab opens.


File Access

Access Control for project: **us-sales** path: **/**

View Rules for: All — Reset filter

Read Permissions (3)

Condition	Condition Details
Property	Roles is one of [Analyst,Manager,Salesperson,Support]
Group	ProDiver Users
User	susanb

Write Permissions (3)

Condition	Condition Details
Property	Function is one of [Analyst,Manager]
Group	ProDiver Users
User	susanb

3. Click add—plus sign (+)—to add a rule row to the **Read Permissions** or **Write Permissions** tables.
4. Set the **Condition** and **Condition Details** (the *who* the rule applies to).
5. Set the *read* and *write* permissions rules—one rule per row, but you can set as many rule rows as necessary.
6. Save the tab.

Setting file access rules at the project root allows those users to see all the directories in the project. Effectively, you are giving *all* users access to *all* directories in the project. It is likely that you will want to hide many of the directories from most users. See Using Inherit from ancestor on the next page.

See also:

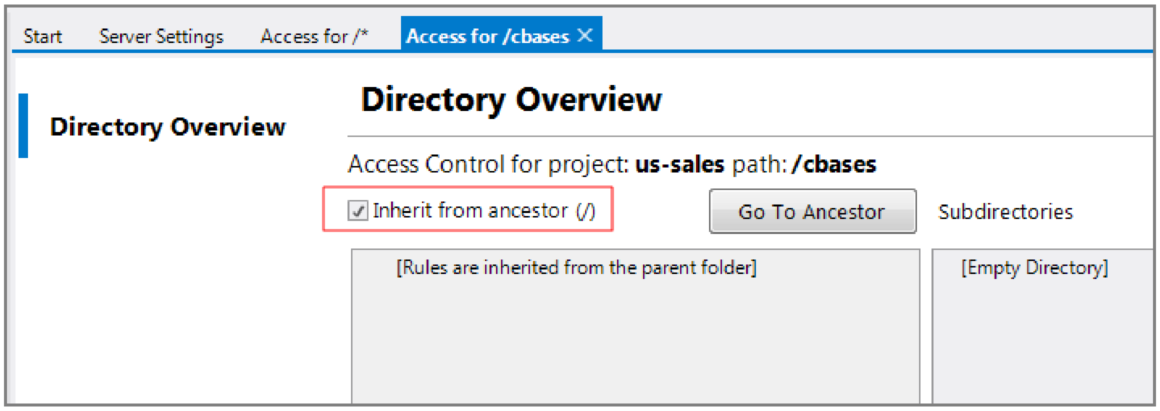
- Configuring Access Control on page 26
- Access Control Overview on page 9
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

Using **Inherit from ancestor**

Controlling access to the folders in a Workbench project is achieved by first granting users *read* and *write* access to the entire project and then restricting access to all but the desired project directories.

To hide a directory from users (for example, as seen in the ProDiver **Open** dialog box), repeat the following steps for each directory where you wish to restrict access:

1. Open the access tab (right-click the directory > **Edit Access Control**)
2. Remove the check mark in the **Inherit from ancestor** box in the **Directory Overview** tab of the **Access for <*directory name*>** tab.



Access Control for project: **us-sales** path: **/cbases**

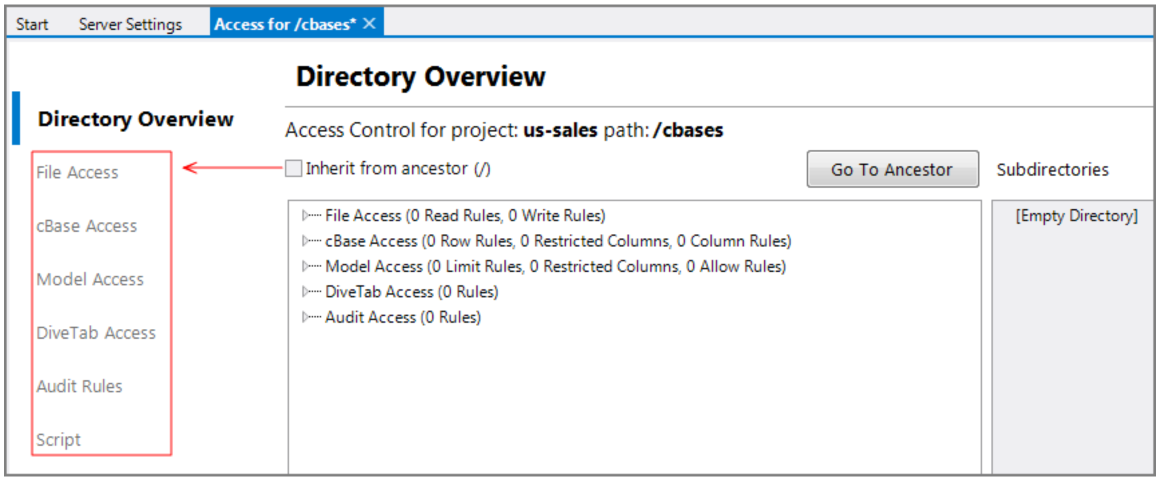
☑ Inherit from ancestor (/)

[Rules are inherited from the parent folder]

[Empty Directory]

NOTE:

- The access tab displays only the **Directory Overview** sub-tab when **Inherit from ancestor** is checked. All the additional sub-tabs and the rules summary appear when the **Inherit from ancestor** check mark is removed as shown here.



Access Control for project: **us-sales** path: **/cbases**

☐ Inherit from ancestor (/)

- File Access (0 Read Rules, 0 Write Rules)
- cBase Access (0 Row Rules, 0 Restricted Columns, 0 Column Rules)
- Model Access (0 Limit Rules, 0 Restricted Columns, 0 Allow Rules)
- DiveTab Access (0 Rules)
- Audit Access (0 Rules)

[Empty Directory]

- To focus the ProDiver **Open** dialog box for a user or group such that it only shows one or two directories, you need to remove this **Inherit from ancestor** check mark for all the other directories you wish to hide.

See also:

- Configuring Access Control on page 26
- Access Control Overview on page 9
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

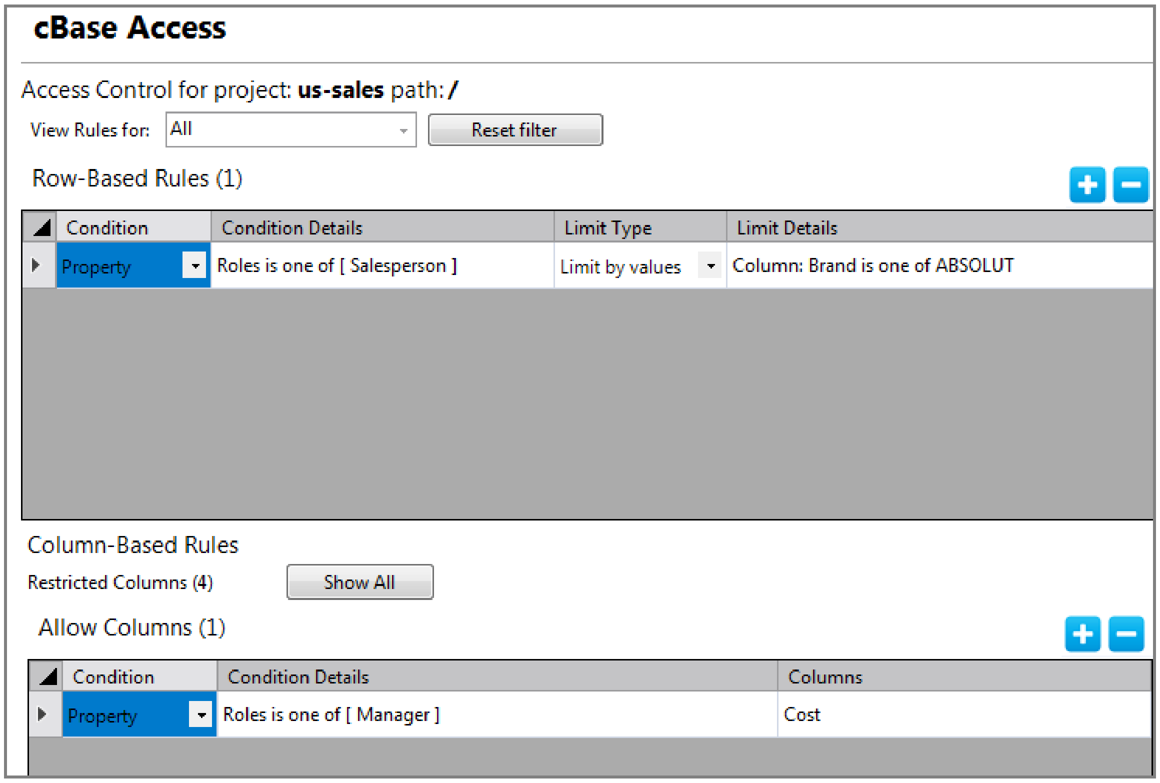
Setting cBase Access

The **cBase Access** sub-tab of the access tab allows you to set *Row-Based* and *Column-Based* access rules for your Workbench project.

To set cBase access:

1. Right-click the project root (or a sub-directory) and click **Edit Access Control** to open the **Access for/** tab.
2. In the **Access for/** tab, click **cBase Access**.

   The **cBase Access** tab opens.


cBase Access

Access Control for project: **us-sales** path: **/**

View Rules for: All

Row-Based Rules (1)

Condition	Condition Details	Limit Type	Limit Details
Property	Roles is one of [Salesperson]	Limit by values	Column: Brand is one of ABSOLUT

Column-Based Rules

Restricted Columns (4)

Allow Columns (1)

Condition	Condition Details	Columns
Property	Roles is one of [Manager]	Cost

3. Click add—plus sign (+)—to add a rule row to the **Row-Based Rules** or **Column-Based Rules** tables.
4. Set the **Condition** and **Condition Details** (the *who* the rule applies to).
5. In the **Row-Based Rules** section, set any *limit by* rules.
6. In the **Column-Based Rules** section, you first choose which columns to restrict (see Using the Restrict Column Editor on page 33) and then set rules in the **Column-Based Rules** table to allow some users or groups access to those restricted columns.
7. Save the tab.

See also:

- Configuring Access Control on page 26
- Access Control Overview on page 9
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

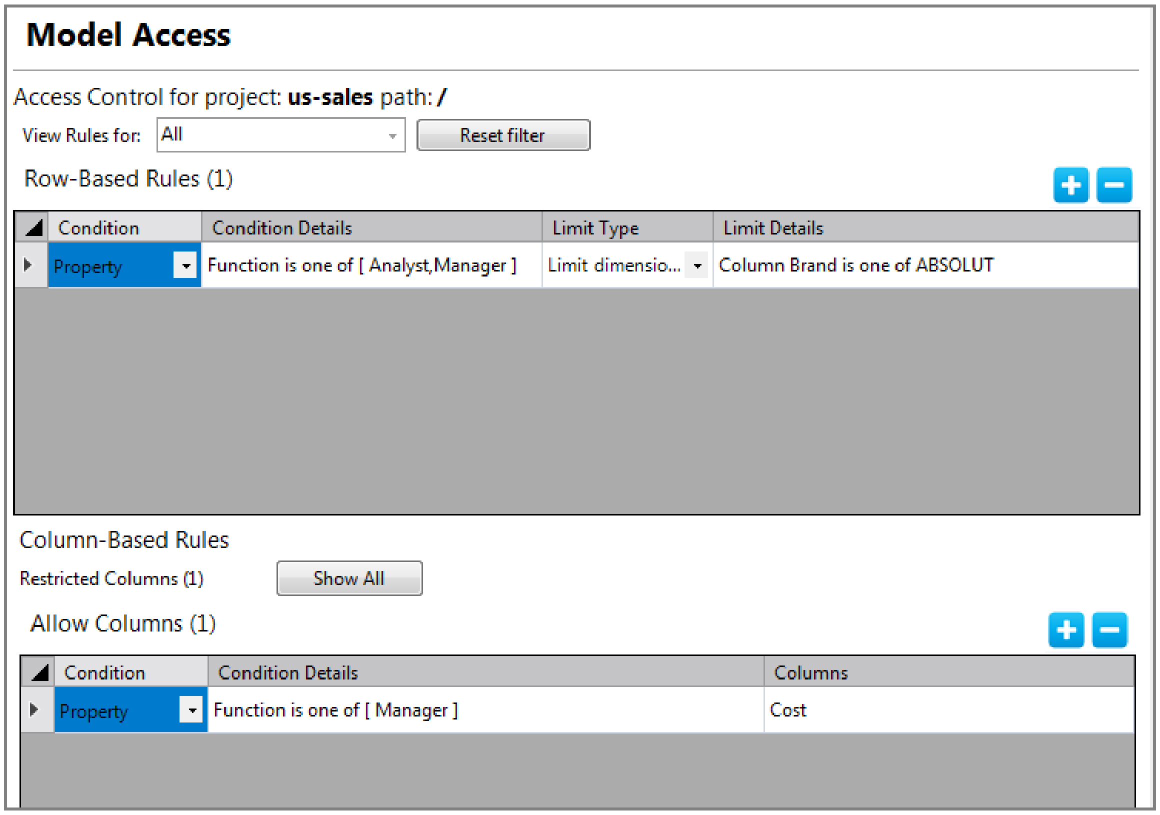
Setting Model Access

The **Model Access** sub-tab of the access tab allows you to set *Row-Based* and *Column-Based* access rules for your Workbench project.

To set model access:

1. Right-click the project root (or a sub-directory) and click **Edit Access Control** to open the **Access for** / tab.
2. In the **Access for** / tab, click **Model Access**.

   The **Model Access** tab opens.



Model Access

Access Control for project: **us-sales** path: **/**

View Rules for: All

Row-Based Rules (1)

Condition	Condition Details	Limit Type	Limit Details
Property	Function is one of [Analyst,Manager]	Limit dimensio...	Column Brand is one of ABSOLUT

Column-Based Rules

Restricted Columns (1)

Allow Columns (1)

Condition	Condition Details	Columns
Property	Function is one of [Manager]	Cost

3. Click add—plus sign (+)—to add a rule row to the **Row-Based Rules** or **Column-Based Rules** tables.
4. Set the **Condition** and **Condition Details** (the *who* the rule applies to).
5. In the **Row-Based Rules** section, set any *limit dimension by* rules.
6. In the **Column-Based Rules** section, you first choose which columns to restrict (see Using the Restrict Column Editor on the next page) and then set rules in the **Column-Based Rules** table to allow some users or groups access to those restricted columns.
7. Save the tab.

See also:

- Configuring Access Control on page 26
- Access Control Overview on page 9
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

Using the Restrict Column Editor

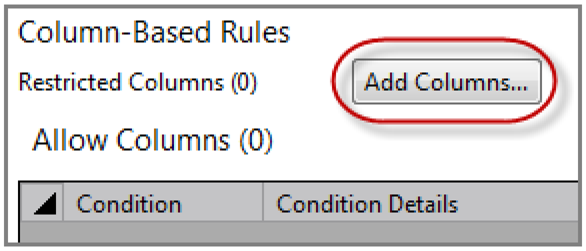
The **Restrict Columns Editor** is used from the **Column-Based Rules** section (for both the **cBase Access** and **Model Access** sub-tabs) to set the columns you wish to restrict users from seeing. This is configured the same in both the **cBase Access** and **Model Access** sub-tabs of the **Access for** tab.

NOTE: The **Restricted Areas** section of the **DiveTab Access** tab uses a similar dialog box (**Restrict Areas Editor**) for restricting access to areas.

This procedure is based on using a cBase example—the same steps apply to setting restricted columns for models and restricted areas for DiveTab.

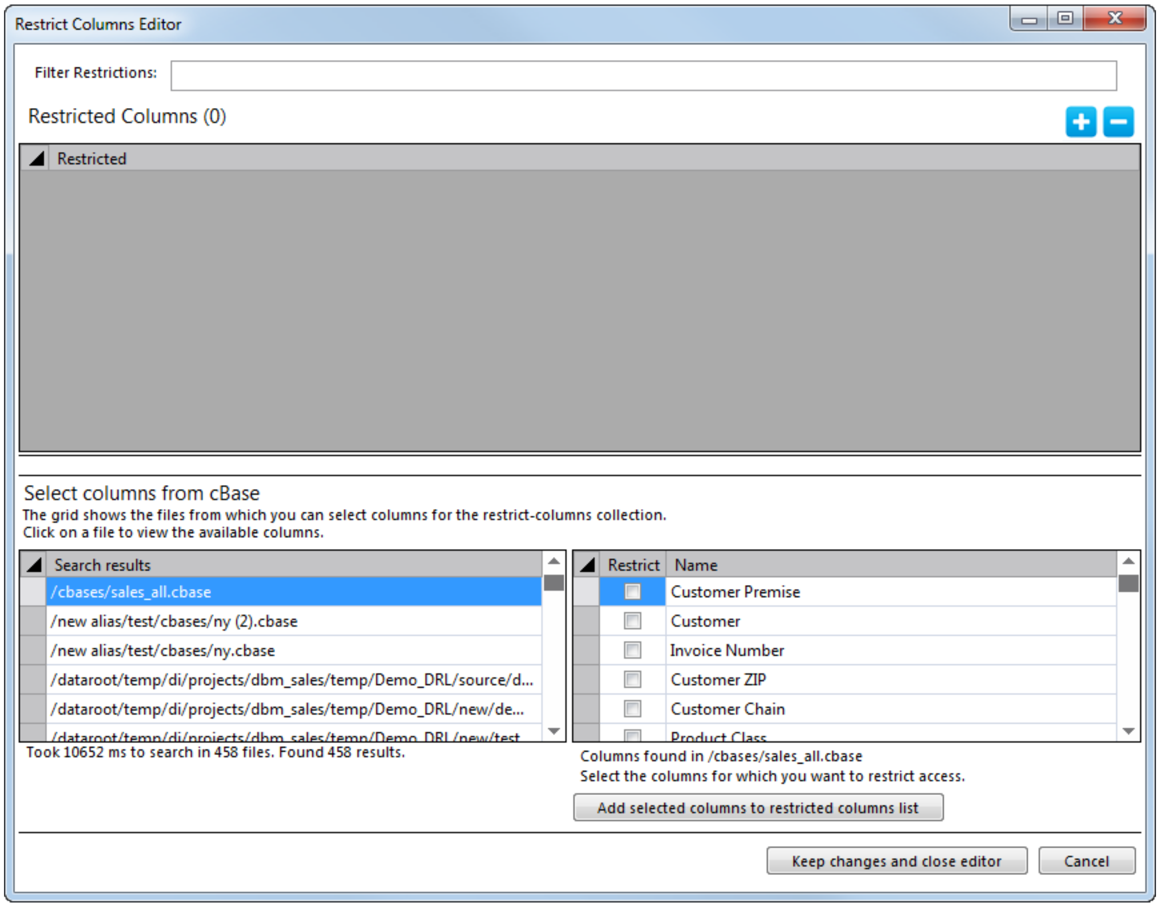
To set columns you want restricted for a cBase:

1. Open an access tab for a project or directory (right-click > **Edit Access Control**).
2. Click **cBase Access**.
3. In the **Column-Based Rules** section, click **Add Columns** next to **Restricted Columns**.



NOTE: If there are any existing settings, the label changes to **Edit** and a preview listing of the restricted columns appears to the right.

The **Restrict Columns Editor** opens.


Restrict Columns Editor

Filter Restrictions:

Restricted Columns (0)

Restricted

Select columns from cBase

The grid shows the files from which you can select columns for the restrict-columns collection.
Click on a file to view the available columns.

Search results
/cbases/sales_all.cbase
/new alias/test/cbases/ny (2).cbase
/new alias/test/cbases/ny.cbase
/dataroot/temp/di/projects/dbm_sales/temp/Demo_DRL/source/d...
/dataroot/temp/di/projects/dbm_sales/temp/Demo_DRL/new/de...
/dataroot/temp/di/projects/dbm_sales/temp/Demo_DRL/new/test

Took 10652 ms to search in 458 files. Found 458 results.

Restrict	Name
☐	Customer Premise
☐	Customer
☐	Invoice Number
☐	Customer ZIP
☐	Customer Chain
☐	Product Class

Columns found in /cbases/sales_all.cbase
Select the columns for which you want to restrict access.

Add selected columns to restricted columns list
Keep changes and close editor
Cancel

4. There are two methods available to add columns:
   - Click **Add** (+) and type the column name in the new row (repeat for as many columns you wish to add).

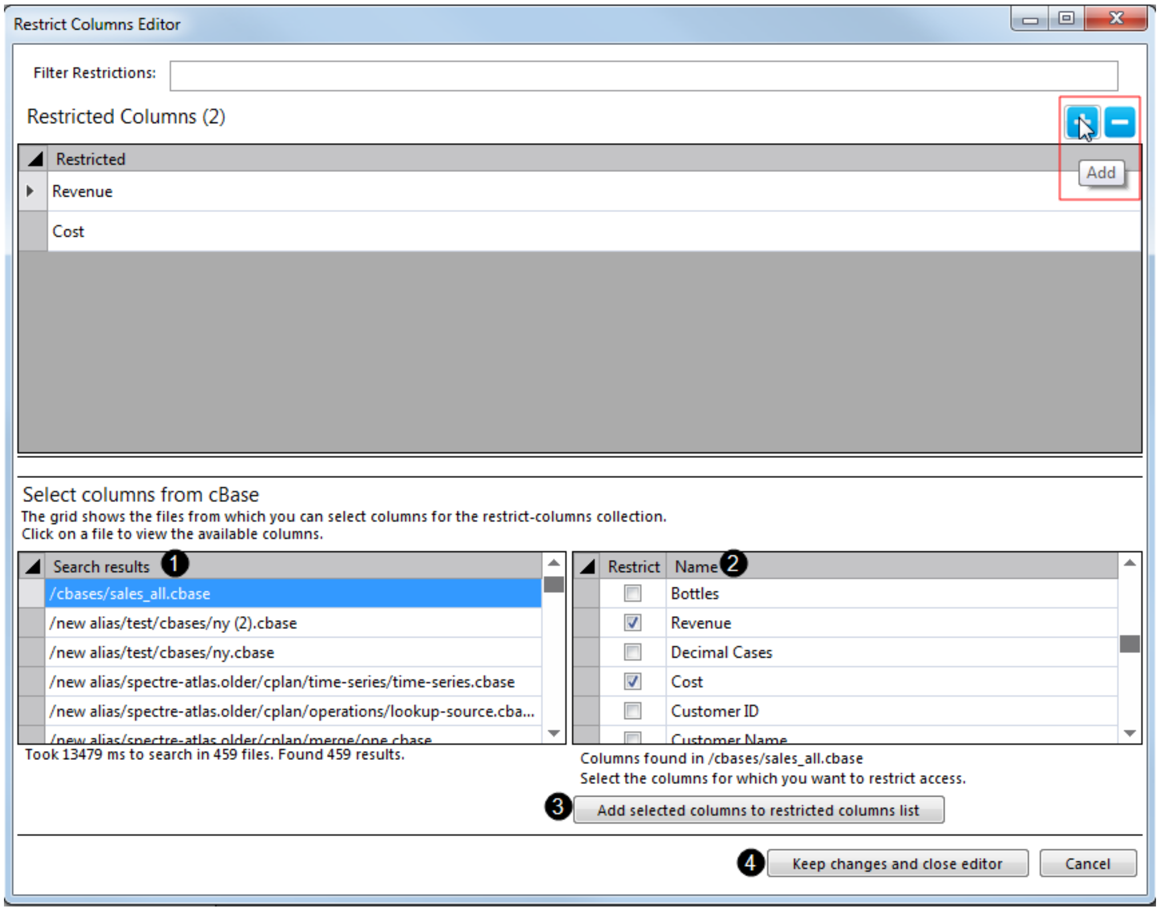
     Or:

   NOTE: Numbers in the sub-steps here refer to numbers in the example graphic shown here.

   a. Select a cBase in the **Search Results** section (1).

   b. Select each columns to restrict in the **Restrict** section (2).

   c. Click **Add selected columns to restricted columns list** (3).
5. Click **Keep changes and close editor** (4).


Restrict Columns Editor

Filter Restrictions:

Restricted Columns (2)

Restricted
Revenue
Cost

Select columns from cBase
The grid shows the files from which you can select columns for the restrict-columns collection.
Click on a file to view the available columns.

Search results
/cbases/sales_all.cbase
/new alias/test/cbases/ny (2).cbase
/new alias/test/cbases/ny.cbase
/new alias/spectre-atlas.older/cplan/time-series/time-series.cbase
/new alias/spectre-atlas.older/cplan/operations/lookup-source.cba...
/new alias/spectre-atlas.older/cplan/merge/one.cbase

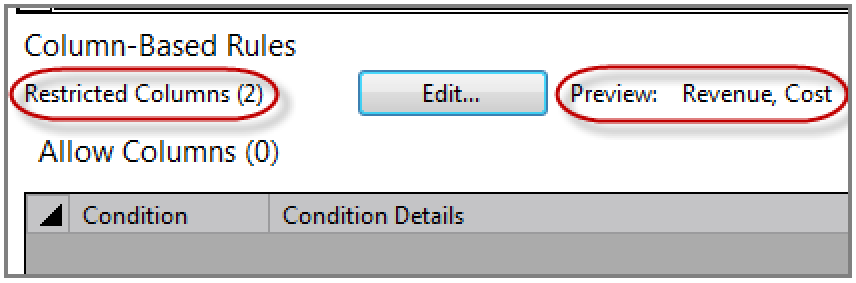
Took 13479 ms to search in 459 files. Found 459 results.

Restrict	Name
☐	Bottles
✓	Revenue
☐	Decimal Cases
✓	Cost
☐	Customer ID
☐	Customer Name

Columns found in /cbases/sales_all.cbase
Select the columns for which you want to restrict access.

Add
1
2
3
Add selected columns to restricted columns list
4
Keep changes and close editor
Cancel

NOTE: A parenthetical number appears next to the **Restricted Columns** area and the column names are listed in the preview area as shown here.


Column-Based Rules
Restricted Columns (2)
Preview: Revenue, Cost
Allow Columns (0)

Condition	Condition Details

Setting DiveTab Access

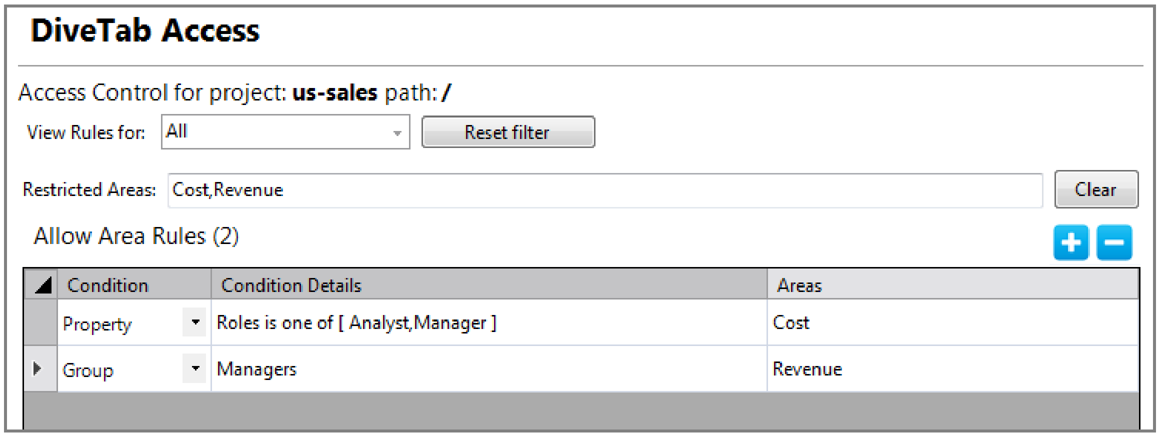
The **DiveTab Access** sub-tab of the access tab allows you to set *Allow Area* access rules for your Workbench DiveTab project.

To set DiveTab access:

1. Right-click the project root (or a sub-directory) and click **Edit Access Control** to open the **Access for /** tab.

2. In the **Access for /** tab, click **DiveTab Access**.

   The **DiveTab Access** tab opens.



DiveTab Access

Access Control for project: **us-sales** path: **/**

View Rules for: All

Restricted Areas: Cost,Revenue

Allow Area Rules (2)

Condition	Condition Details	Areas
Property	Roles is one of [Analyst,Manager]	Cost
Group	Managers	Revenue

3. Click add—plus sign (+)—to add a rule row to the **Allow Area Rules** table.
4. Set the **Condition** and **Condition Details** (the *who* the rule applies to).
5. Set areas to restrict in the **Restricted Areas** text box (separated by commas with no spaces) and then set rules in the **Allow Area Rules** table to allow some users or groups access to those restricted areas.
6. Save the tab.

See also:

- Configuring Access Control on page 26
- Access Control Overview on page 9
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

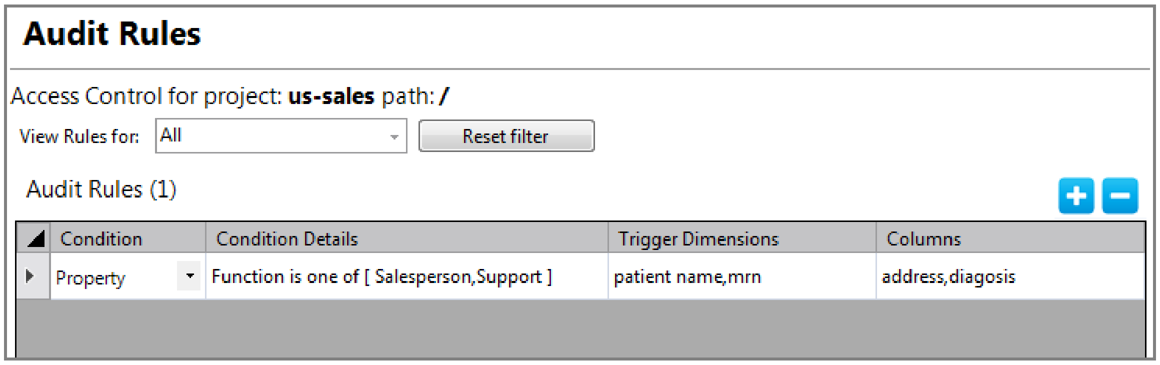
Setting Audit Rules

The **Audit Rules** sub-tab of the access tab allows you to set *Audit* access rules for your Workbench project.

To set audit rules

1. Right-click the project root (or a sub-directory) and click **Edit Access Control** to open the **Access for /** tab.
2. In the **Access for /** tab, click **Audit Rules**.

   The **Audit Rules** tab opens.


Audit Rules

Access Control for project: **us-sales** path: **/**

View Rules for: All

Reset filter

Audit Rules (1)

Condition	Condition Details	Trigger Dimensions	Columns
Property	Function is one of [Salesperson,Support]	patient name,mrn	address,diagosis

3. Set the **Condition** and **Condition Details** (the *who* the rule applies to).
4. Set audit rules by listing trigger dimensions and the columns viewed to cause a log entry (separated by commas with no spaces).
5. Save the tab.

   NOTE: In order to have the column's accessed values appear in the audit log, you must list the columns in both the **Trigger Dimensions** and **Columns** sections for each audit rule.

See also:

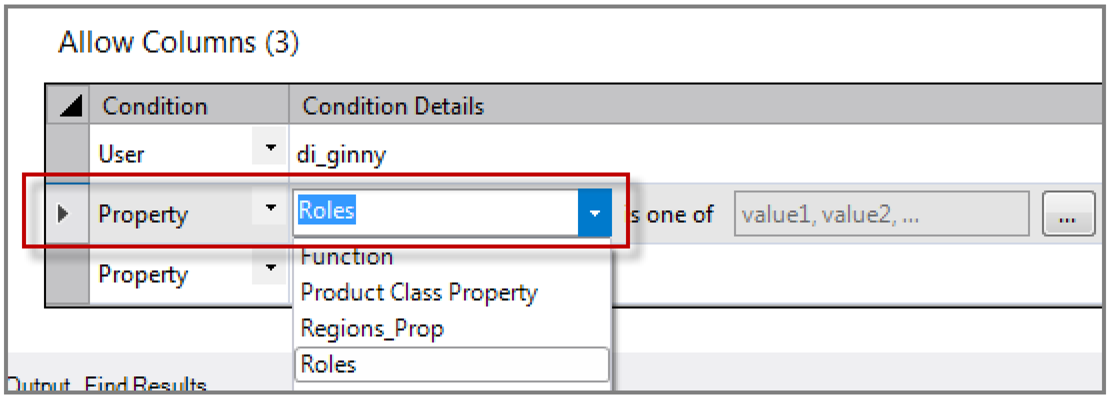
- Audit Logging on page 4
- Configuring Access Control on page 26
- Access Control Overview on page 9
- About Access Control on page 11
- Access Control Conditions on page 19
- Access Control Effects on page 22
- Underlying Access Control Script on page 40

Using the Select Property Values Dialog Box

For any rule set with the condition of **Property**, you use the **Select Property Values** dialog box to assign one or more values.

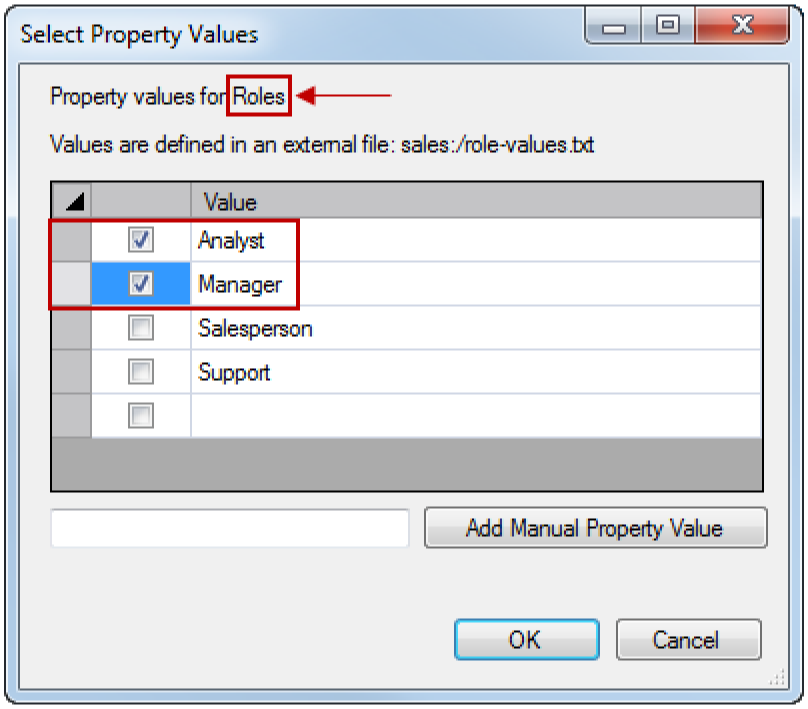
To assign values for a property condition:

1. Select **Property** for the rule condition and choose a condition detail from the list as shown here.



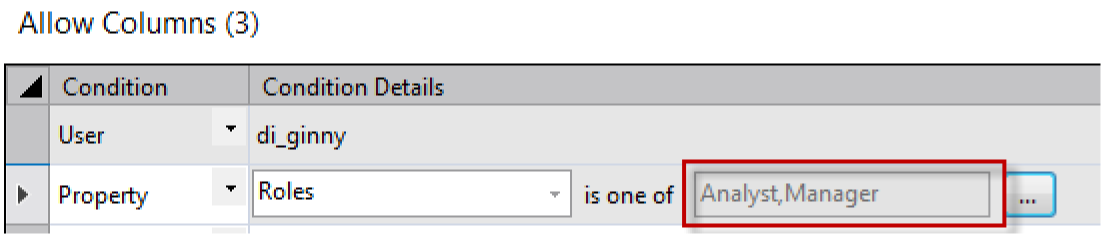
2. Click browse next to the values field.

   The **Select Property Values** dialog box appears for the selected condition detail.


Select Property Values
Property values for Roles
Values are defined in an external file: sales:/role-values.txt
Value
✓ Analyst
✓ Manager
Salesperson
Support
Add Manual Property Value
OK
Cancel

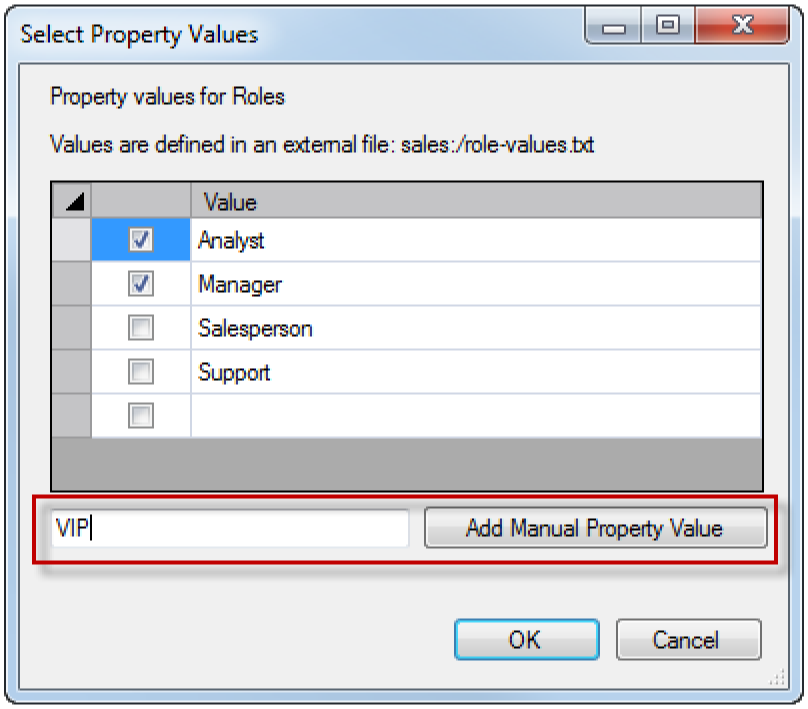
3. Click the check box for each of the property values you want assigned to this rule and click **OK**.

   The values are assigned to this rule as shown here.



If you find a need for a property value that does not yet exist, you can use the **Add Manual Property Value** feature to insert the new value while creating the rule. Make sure to add the value later.

1. In the **Select Property Values** dialog box, type a new value and click **Add Manual Property Value**.



Select Property Values

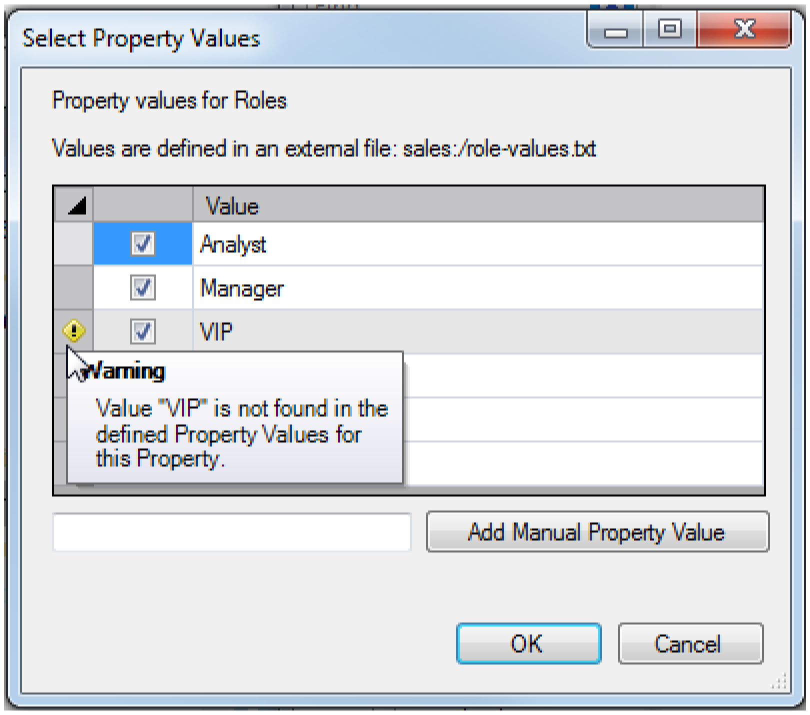
Property values for Roles

Values are defined in an external file: sales:/role-values.txt

	Value
✓	Analyst
✓	Manager
☐	Salesperson
☐	Support
☐	

VIP

The value is added to the value listing as shown here. Note the warning icon reminding you that this value is not found in the defined values of this property.


Select Property Values

Property values for Roles

Values are defined in an external file: sales:/role-values.txt

	Value
✓	Analyst
✓	Manager
✓	VIP

Warning

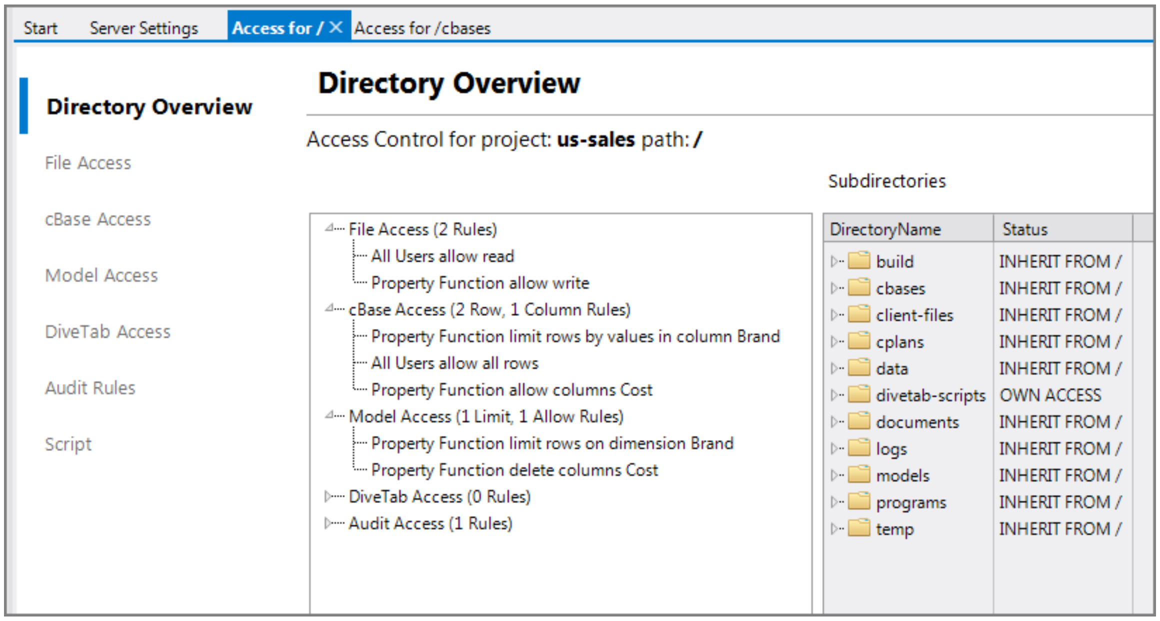
Value "VIP" is not found in the defined Property Values for this Property.

2. Be sure that this property value is added to the property as defined in **Server Settings** > **Properties**.

Underlying Access Control Script

As you set the access rules, an underlying script is generated. This script is available on the **Script** sub-tab of the access tab. This script tab is interactive and you can make edits directly in the script. As you work in either the access tab interface or the script tab, all changes made in one place appear in the other.

An example screen of a directory overview tab is shown here, which shows a summary of all access rules for project **us-sales**.



Directory Overview

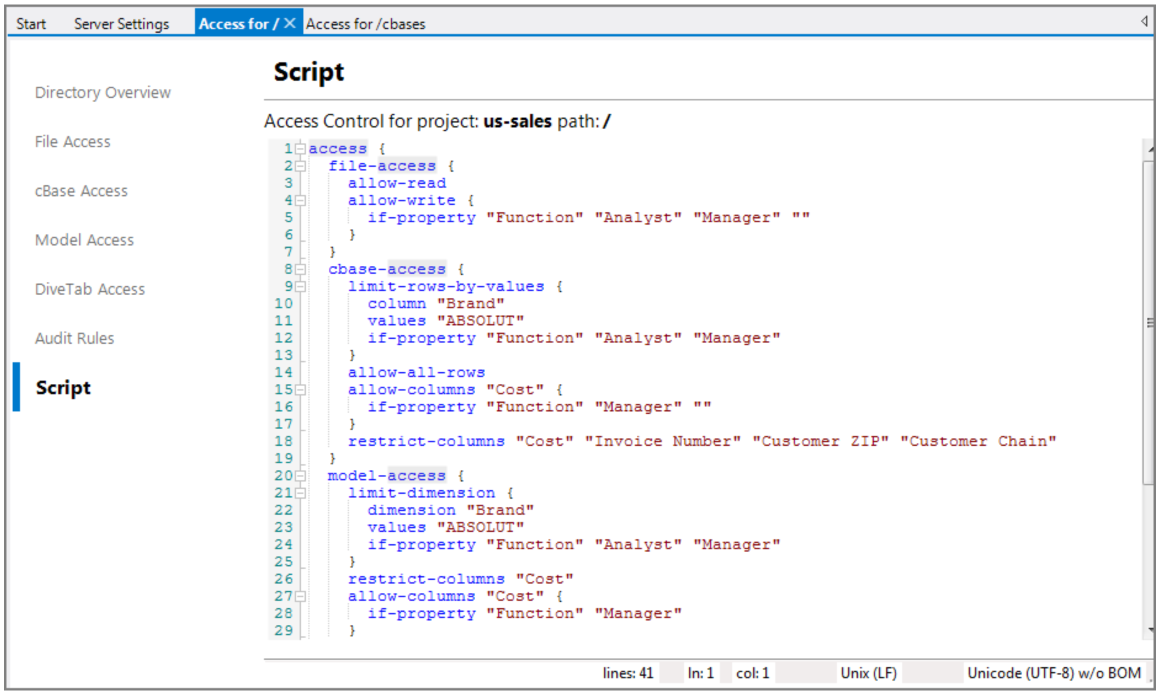
Access Control for project: **us-sales** path: **/**

- File Access (2 Rules)
  - All Users allow read
  - Property Function allow write
- cBase Access (2 Row, 1 Column Rules)
  - Property Function limit rows by values in column Brand
  - All Users allow all rows
  - Property Function allow columns Cost
- Model Access (1 Limit, 1 Allow Rules)
  - Property Function limit rows on dimension Brand
  - Property Function delete columns Cost
- DiveTab Access (0 Rules)
- Audit Access (1 Rules)

Subdirectories

DirectoryName	Status
build	INHERIT FROM /
cbases	INHERIT FROM /
client-files	INHERIT FROM /
cplans	INHERIT FROM /
data	INHERIT FROM /
divetab-scripts	OWN ACCESS
documents	INHERIT FROM /
logs	INHERIT FROM /
models	INHERIT FROM /
programs	INHERIT FROM /
temp	INHERIT FROM /

The resultant script tab is shown here.



Script

Access Control for project: **us-sales** path: **/**

```
access {
  file-access {
    allow-read
    allow-write {
      if-property "Function" "Analyst" "Manager" ""
    }
  }
  cbase-access {
    limit-rows-by-values {
      column "Brand"
      values "ABSOLUT"
      if-property "Function" "Analyst" "Manager"
    }
    allow-all-rows
    allow-columns "Cost" {
      if-property "Function" "Manager" ""
    }
    restrict-columns "Cost" "Invoice Number" "Customer ZIP" "Customer Chain"
  }
  model-access {
    limit-dimension {
      dimension "Brand"
      values "ABSOLUT"
      if-property "Function" "Analyst" "Manager"
    }
    restrict-columns "Cost"
    allow-columns "Cost" {
      if-property "Function" "Manager"
    }
```

lines: 41 ln: 1 col: 1 Unix (LF) Unicode (UTF-8) w/o BOM

Click the **Script example** link below to see the full script.

Script example

```
access {
  file-access {
    allow-read
    allow-write {
      if-property "Function" "Analyst" "Manager" ""
    }
  }
  cbase-access {
    limit-rows-by-values {
      column "Brand"
      values "ABSOLUT"
      if-property "Function" "Analyst" "Manager"
    }
    allow-all-rows
    allow-columns "Cost" {
      if-property "Function" "Manager" ""
    }
    restrict-columns "Cost" "Invoice Number" "Customer ZIP"
"Customer Chain"
  }
  model-access {
    limit-dimension {
      dimension "Brand"
      values "ABSOLUT"
      if-property "Function" "Analyst" "Manager"
    }
    restrict-columns "Cost"
    allow-columns "Cost" {
      if-property "Function" "Manager"
    }
  }
  audit-rules {
    audit {
      trigger "patient name"
      trigger "mrn"
      column "address"
      column "diagosis"
      if-property "Function" "Salesperson" "Support"
    }
  }
}
```

For more information about working with the script syntax, see:

- Access Control File Samples on the next page
- Access Control File Code Block on page 48

- Access Control File Tags on page 52

See also:

- About Access Control on page 11
- Access Control Process on page 14
- Access Control Categories on page 16

Access Control File Samples

The **Access Control** file must start with **access { }** to set the file type. Once the file type is set, the Workbench text editor can offer suggested tags. Place the cursor where you want to add a function and press **Ctrl+Space** and a pop-up window displays possible tags.

If no access control file is set, or if you save an entirely empty file as the access control, then the access control is set to inherit from the parent directory. If this is the top-level of the project, then all access is denied.

NOTE: Administrator users are not subject to access control restrictions. They are however, subject to any non-conditional audit rules.

Some examples of access control files are shown below.

Default Access Example

In this example, the default setting allows this folder to have the same access control rules as the parent folder.

```
access {
  inherit-from-parent-folder
}
```

File Access Example

In this example, all users are allowed to read files in this folder, but only user *Burt* can write to them.

```
access {
  file-access {
    allow-read
    allow-write {
     if-user "Burt"
    }
  }
}
```

cBase Access Examples

In this example, the property *Manager* is used to allow users with this property to view the

Profit column.

```
access {
  cbase-access {
    restrict-columns "Profit"
    allow-columns "Profit" {
      if-property "Manager"
    }
  }
}
```

NOTE: You can use only one **restrict-columns** tag within a **cbase-acceess** tag. Multiple **allow-columns** tags are allowed so you can allow back in columns for specific users, groups, or those with a certain property applied.

In this example, an expression is used to create a union of two different rules.

```
cbase-access {
    limit-rows-by-filter {
      filter ```
      user_property_contains("pfm-fac-id", value("Facility
ID"))    or
      user_property_contains("pfm-gac-dep-id", value
("Facility Code-Dept ID"))
      ```
    }
  }
```

In this example, an expression is used to simulate wild card values such as 10* and 20*.

```
cbase-access {
    limit-rows-by-filter {
      filter `is_in(substr(value("Sales Manager"), 1, 2),
"10", "20")`
    }
  }
```

The following script shows two alternatives for controlling access to the data by user.

```
  // To control access to rows by user, use the current_user
function with a filter limit...
  cbase-access {
    limit-rows-by-filter {
      filter `value("Dimension") = current_user()`
    }
  }
  // ... or use the username limit instead
  //cbase-access {
```
```

```
//  limit-rows-by-username-column {
//    column "Dimension"
//  }
//}
```

Model Access Example

In this example, the sales region data is limited for the *Managers* group. This is for a classic Model in a Workbench project.

```
access {
  model-access {
    limit-dimension {
      if-group "Managers"
      dimension "Sales Region"
      values "New England" "Mid-Atlantic" "South East"
    }
}
```

Audit Rules Example

In this example, the audit block contains two trigger declarations, and two additional columns to log, in addition to the trigger dimensions.

```
access {
  audit-rules {
    audit {
      trigger "Patient Name"
      trigger "MRN"
      column "Address"
      column "Diagnosis"
    }
  }
}
```

Project Access Example

Project access, at the root level, defaults to **inherit-from-parent-folder**. When you start refining the access using **Tools** > **Server Settings** > **Projects** > **Project Access**, and grant access to a non-administrative user or group, or set `allow-all-users`, the access control data for the project access is stored in a text file at:

```
<dataroot>\config\projects\<projectname>\project-access-
config.sdl
```

This file is in SDL format, not tab delimited. Occasionally there may be a desire to update the contents with an Integrator or other type of script.

In the following example, access to the project is limited to two users and one group.

```
project-access {
  allow-user "frank"
  allow-user "lisa"
  allow-group "Managers"
}
```

See also:

- About Access Control on page 11
- Access Control File Samples on page 43
- Access Control Model Sample below
- Access Control File Code Block on page 48
- Access Control File Tags on page 52

Access Control Model Sample

Model files not in a project (but in the 6.4 DiveLine namespace) retain their ACLs as setup in DI-Config.

Models in a 7.0 project, whether created new or aliased in, have their access defined in Workbench.

Access control for Models in Workbench projects could resemble the following:

```
  model-access {

    // The class limit, for the "Managers" group only
    limit-dimension {
      if-group "Managers"
      dimension "Sales Region"
      values "South*" "Not-appearing-in-this-film" "North"
    }

    // Same as $groups outside projects
    limit-dimension-by-groups {
      dimension "Product Family"
    }

    // Same as $user outside projects
    limit-dimension-by-username {
      dimension "salesperson-uid"
    }

    // New!
    limit-dimension-by-property {
      dimension "Product Name"
      property "Products"
    }

    // New!
    restrict-columns "Revenue" "Profit"
    allow-columns "Revenue" "Profit" {
      if-group "Managers"

    restrict-columns "Revenue" "Profit"
    allow-columns "Revenue" "Profit" {
      if-property "Role" "Manager"

    restrict-columns "Revenue" "Profit"
    allow-columns "Revenue" "Profit" {
      if-user "CEO"
    }
```

NOTE: The `delete-columns` tag has been deprecated; use the new `restrict-columns` and `allow-columns` tags instead.

```
    // Deprecated
    delete-columns "Cost" { if-user "someone" }
    delete-columns "Revenue" { if-group "group B" }
    delete-columns "Sensitive" { if-property "Sensitivity"
"0" "1" "2" "3" }
    delete-columns "Units" {
      if-group "A group"
    }
  }
}
```

NOTE: Previous versions using Model ACLs for security could not OR conditions. Access control for 7.0 projects and cBases supports the union of access control rules when using **limit-rows-by-filter,** which accepts an arbitrary Spectre expression for allowing or disallowing each row in a cBase.

See also:

- About Access Control on page 11
- Access Control Model Sample on the previous page
- Access Control File Samples on page 43

- Access Control File Code Block below
- Access Control File Tags on page 52

Access Control File Code Block

Below is a code block displaying all of the possible tags that you may use in an access control file (*access*).

NOTE: You can use only one of the `if` tags per parent tag. They are all listed here for completeness and ease of use for copy/paste.

Access

```
access {
    inherit-from-parent-folder
}
```

File Access

```
file-access {
    allow-read {
      if-group <group>
      if-property <property> <value list>
      if-user <user>
  }
    allow-write{
      if-group <group>
      if-property <property> <value list>
      if-user <user>
  }
}
```

cBase Access

```
cbase-access {
    restrict-columns <columns>
    allow-columns <columns> {
      if-group <group>
      if-property <property> <value list>
      if-user <user>
    }
    allow-all-rows {
      if-group <group>
      if-property <property> <value list>
      if-user <user>
    }
    limit-rows-by-filter {
      filter <expression>
      if-group <group>
      if-property <property> <value list>
      if-user <user>
    }
    limit-rows-by-property {
      column <column>
      property <property>
      if-group <group>
      if-property <property> <value list>
      if-user <user>
    }
    limit-rows-by-username-column {
      column <column>
      if-group <group>
      if-property <property> <value list>
      if-user <user>
    }
    limit-rows-by-values {
      column <column>
      values <values>
      if-group <group>
      if-property <property> <value list>
      if-user <user>
    }
}
```

DiveTab Access

```
divetab-access {
    restrict-areas <id> <id> <id>
    allow-area <id> {
      if-group <group>
      if-property <property> <value list>
      if-user <user>
    }
}
```

Model Access

```
model-access {
  restrict-columns <columns>
  allow-columns <columns> {
    if-group <group>
    if-property <property> <value list>
    if-user <user>
  }
  limit-dimension {
    dimension <dimension>
    values <value list>
    if-group <group>
    if-property <property> <value list>
    if-user <user>
  }
  limit-dimension-by-property {
    dimension <dimension>
    property <property>
    if-group <group>
    if-property <property> <value list>
    if-user <user>
  }
  limit-dimension-by-username {
    dimension <dimension>
    if-group <group>
    if-property <property> <value list>
    if-user <user>
  }
  limit-dimension-by-groups {
    dimension <dimension>
    if-group <group>
    if-property <property> <value list>
    if-user <user>
  }
```

NOTE: The following `delete-columns` tag has been deprecated; use the new `restrict-columns` and `allow-columns` tags instead. The old syntax still works and is available for migration situations. Workbench will suggest updating the model access when it detects the deprecated tags in use, or discrepancies between the client and server versions. Attempts to mix the old and new tags result in DiveLine denying access to the model.

```
  delete-columns <column list> {
    if-group <group>
    if-property <property> <value list>
    if-user <user>
  }
```

Audit Rules

```
audit-rules {
  audit {
      trigger <dimension>
      trigger <dimension>
      column <log column>
      column <log column>
      if-group <group>
      if-property <property> <value list>
      if-user <user>
  }
}
```

Project Access

```
project-access {
  allow-user <user>
  allow-user <user>
  allow-group <group>
}
project-access {
  allow-all-users
}
```

NOTE: For `<value list>`, list the values as separate strings. For example:

```
values "first value" "second value" "third value" "etc"
```

See also:

- About Access Control on page 11
- Access Control File Code Block on page 48
- Access Control File Samples on page 43
- Access Control Model Sample on page 46
- Access Control File Tags on the next page
- Audit Logging on page 4

Access Control File Tags

An access control file is defined using an **access** tag with optional tags between an open **{** and closed **}** brackets. Access control files are set at either the project or folder level with the right-click > **Edit Access Control** command and click the **Scripts** sub-tab to view the underlying code that the GUI creates. Examples in the table would apply to the project or folder where this access control file resides.

Available tags within the Access Control File

Tag	Example	Notes

access	access {
	}
		Top-level tag that sets the file type (required). Defaults to no access if there are no optional tags used.

inherit-from-parent-folder	access {
	inherit-from-parent-folder
	}
		Second level tag when no access has been defined. No other tags should be present. This is the same as a directory with no access file.
		NOTE: This is the default access set for all folders in the project except the root folder of the project.

file-access	file-access {
	<third-level tags>
	}
		Second-level tag that can allow or restrict read/write access.

allow-read	file-access {
	allow-read
	}
		Allows read-access to project or folder. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

allow-write	file-access {
	allow-write
	}
		Allows write-access to project or folder. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

Tag	Example	Notes

if-group
if-property
if-user
	file-access {
	   allow-write {
	      if-group "Managers"
	   }
	}
		Fourth-level tags used to allow/restrict read/write access by group, property, or user. One **if** statement per block is allowed, though you can have multiple blocks.

cbase-access	cbase-access {
	   <third-level tags>
	}
		Second-level tag used to allow or restrict access to cBase files.

allow-all-rows	cbase-access {
	   allow-all-rows
	}
		Allows access to all rows. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

Tag	Example	Notes

limit-rows-by-filter	cbase-access {
	  limit-rows-by-filter {
	    <spectre expression>
	  }
	}
	cbase-access {
	  limit-rows-by-filter {
	    filter `value("Username") = current_user()`
	  }
	}
	cbase-access {
	  limit-rows-by-filter {
	    filter
	```
	      user_property_contains("A", value("Dimension A")) or user_property_contains("B", value("Dimension B"))
	```
	  }
	}
	cbase-access {
	  limit-rows-by-filter {
	    filter `regexp(value("Sales Manager"), "^[12]0")`
	  }
	}
		Limits access to rows by an expression. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.
		NOTE: When a custom filter expression includes the **user-property_contains()** function, and a user lacks any assignments for a particular user property (when assignments are made in an external file), the function returns false when finding an empty set.
		Use **limit-rows-by-filter** to effect the union of access control rules; any Spectre expression can be used for allowing or disallowing each row in the cBase.

limit-rows-by-property	cbase-access {
	  limit-rows-by-property {
	    column "Sales Manager"
	    property "high-clearance"
	  }
	}
		Limits access to rows by property. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.
		The following deprecated tags are mapped to limit-rows-by-property: limit-rows-by-property-value, limit-rows-by-property-values and limit-rows-by-value.

Tag	Example	Notes

limit-rows-by-username-column	cbase-access {
	  limit-rows-by-username-column {
	    column "Sales Manager"
	  }
	}
		Limits access to rows by username and column. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

limit-rows-by-values	cbase-access {
	  limit-rows-by-values {
	    column "Sales Manager"
	    values "10" "20"
	  }
	}
		Limits access to rows by column and values. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

column	column "X"	Provides the column name for the limit tag.
values	values "A" "B" "C"	Provides values for the column for the limit tag.

if-group
if-property
if-user
	cbase-access {
	  allow-columns "Profit" "Cost" {
	    if-property "high-clearance" "yes"
	  }
	}
		Fourth-level tags used to allow/limit access to columns by group, property, or user. One **if** statement per block is allowed, though you can have multiple blocks.

restrict-columns	cbase-access {
	  restrict-columns "Profit" "Expenses"
	}
		Restricts access to named columns. Only one allowed per **cbase-access** tag, though you can list multiple columns.

Tag	Example	Notes

allow-columns	cbase-access {
	  allow-columns "Supplier" "Customer" "Price"
	}
	cbase-access {
	  restrict-columns "revenue"
	  allow-columns "revenue" {
	    if property "can-see-revenue" "yes" "executive"
	  }
	}
		Allows access to named columns. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first. Multiple **allow-columns** tags are allowed so you can allow back in columns for specific users, groups, or those with a certain property applied.

divetab-access	divetab-access {
	  <third-level tags>
	}
		Second-level tag that can restrict or allow access to areas and ids within a DiveTab project.

restrict-areas	divetab-access {
	  restrict-areas "Presentations" "Export"
	}
		Restricts access to the listed areas.

allow-area	divetab-access {
	  allow-area "Supplier Overview"
	}
		Allows access to the listed id. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

if-group
if-property
if-user
	divetab-access {
	  allow-area "Presentations" {
	    if-property "Sales Force" "yes"
	  }
	}
		Fourth-level tags used to allow access to areas by group, property, or user. One **if** statement per block is allowed, though you can have multiple blocks.

model-access	model-access {
	  <third-level-tags>
	}
		Second-level tag used to allow or restrict access to classic Model files.

Tag	Example	Notes

limit-dimension	model-access {
	  limit-dimension {
	    dimension "X"
	    values "A" "B" "C"
	  }
	}
		Limits access to the listed dimension values. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

dimension	dimension "X"	Provides the dimension name for the limit tag.
values	values "A" "B" "C"	Provides values for the dimension for the limit tag.

limit-dimension-by-property	model-access {
	  limit-dimension-by-property {
	    dimension "Y"
	    property "Allowed Y Values"
	  }
	}
		Limits access to dimension and values by property. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

limit-dimension-by-username	model-access {
	  limit-dimension-by-username {
	    dimension "Username"
	  }
	}
		Limits access to dimension and values by user. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

limit-dimension-by-groups	model-access {
	  limit-dimension-by-groups {
	    dimension "Group"
	  }
	}
		Limits access to dimension and values by groups. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

restrict-columns	model-access {
	  restrict-columns "Profit" "Expenses"
	}
		Restricts access to named columns. Only one allowed per **model-access** tag, though you can list multiple columns.

Tag	Example	Notes

allow-columns	cbase-access {
	   allow-columns "Supplier" "Customer" "Price"
	}
	model-access {
	   restrict-columns "revenue"
	   allow-columns "revenue" {
	      if property "can-see-revenue" "yes" "executive"
	   }
	}
		Allows access to named columns. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first. Multiple **allow-columns** tags are allowed so you can allow back in columns for specific users, groups, or those with a certain property applied.

if-group
if-property
if-user
	model-access {
	   limit-dimension {
	      dimension "X"
	      values "A" "B" "C"
	      if-user "jdoe"
	   }
	}
		Fourth-level tags used to limit access by user, group, or property. One **if** statement per block is allowed, though you can have multiple blocks.

delete-columns
(Deprecated, use the **restrict-columns** and **allow-columns** tags instead.)
	model-access {
	   delete-columns "R" "S" "T"
	}
		Deletes the named columns from view. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

audit-rules	audit-rules {
	   <audit blocks>
	}
		Second-level tag to define auditing rules.

audit	audit-rules {
	   audit {
	      trigger "profit"
	   }
	   audit {
	      trigger "bonuses"
	   }
	}
		Specifies any number of audit triggers and additional columns to be captured. Multiple audit blocks are allowed. Accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

Tag	Example	Notes

trigger	audit-rules {
	  audit {
	    trigger "MRN"
	    trigger "diagnosis"
	    column "employee number"
	  }
	}
		Defines the dimensions to trigger an audit action. Multiple triggers can be listed in each audit block. Each block accepts one **if** condition tag. If no condition tag, this tag applies to all users. Conditions are applied first.

column	audit-rules {
	  audit {
	    trigger "Patient Account Number"
	    trigger "Patient Name"
	    trigger "MRN"
	    trigger "Encounter ID"
	    column "Column Name"
	    column "Value"
	    column "Username"
	    column "Access Date Time"
	    column "Application"
	  }
	}
		Specifies the columns to be captured in the audit column log files .Multiple columns can be listed in each audit block.

if-group
if-property
if-user
	audit-rules {
	  audit {
	    trigger "Bonuses"
	    column "Employee Name"
	    if-property "Manager" "no"
	  }
	}
		Fourth-level tags used to qualify audits by group, property, or user. One **if** statement per block is allowed, though you can have multiple blocks.

project-access	project-access {
	  allow-all-users
	}
		Top level tag created anytime a non-administrative user or group has been granted access to a project.
		NOTE: This configuration is maintained behind the scenes via **Server Settings** > **Projects.**

Tag	Example	Notes

allow-user
allow-group
allow-all-users
	project-access {
	    allow-group "managers"
	}
		Tags to qualify project access by user or group.

NOTE:

> Rules which include a conditional tag (**if-group, if-property**, **if-user**) are considered **conditional rules**. If no condition is set, then the rule applies to all users.
> When looking at the access file, before making access decisions, DiveLine will scan all the conditional rules and test each to see if it applies to the current user attempting to access data. If the rule does not pass the condition, the rule is removed from consideration—it would be as if the rule were not present at all.
> Only the rules with satisfied conditions, or no conditions at all, are processed to determine effective access.

See also:

- About Access Control on page 11
- Access Control File Tags on page 52
- Access Control File Samples on page 43
- Access Control Model Sample on page 46
- Access Control File Code Block on page 48

How to Create Column Level Security

How would you implement column level security? For example, if you do not want the sales reps to see the profit column.

First, choose which columns in the cBase are sensitive, and restrict them with **restrict-columns**.

NOTE: You can use only one **restrict-columns** tag within a **cbase-acceess** tag. Multiple **allow-columns** tags are allowed so you can allow back in columns for specific users, groups, or those with a certain property applied.

For example, here three columns are restricted:

```
cbase-access {
    restrict-columns "Profit" "Secret Number" "Password"
    ...
```

As a second step, add rules which allow access to the columns to certain people, using a condition to specify who gets each column. For example, if the *Role* property value is either *Manager* or *CEO*:

```
    if-property "Role" "Manager" "CEO" {
      allow-columns "Profit"
    }
```
Or if the user is *Steve*:

```
    if-user "Steve" {
      allow-columns "Profit"
    }
    ...
```
See also:

- Access Control File Code Block on page 48

How to Exclude Access to Values

Sometimes you need to exclude a short list of values, as opposed to creating a long list of values to allow the user to see. This can be achieved with a filter expression in the **access** block. For example:

```
access {
   file-access {
     allow-read
   }
   cbase-access {
     limit-rows-by-filter {
        filter `not(user_property_contains("property name",
value("column name")))`
     }
   }
}
```
This assumes you are storing a list of values (in the cBase's *column name*) to exclude as a User Property (*property name* in this example). The filter expression is run on each record in the cBase, keeping each row where true is returned.

Access Control Files on Server

Suppose you are doing development work on a server. This includes defining the access control for users, groups, and properties, plus the use of external files. What files would you need to copy from the development server to the production server to keep the same access control settings?

The rules themselves are in the project, under the *.project* folder as the file *project.access* and the folder *access*. That is, look for:

```
    di/projects/<project name>/.project/project.access
```

```
di/projects/<project-name>/.project/access
```

The definition of the properties are global to the entire dataroot, and are located at:

```
di/solution/dl-dataroot/config/user-properties-
config.sdl
```

Assignments of property values, for each property, are either

(a) Maintained by Workbench, and thus saved at:
`di/solution/dl-dataroot/config/user-properties-values.txt`
(b) Maintained externally, and thus saved into the text file the property configuration names, which will be inside one of the projects.

See also: Access Control Overview on page 9.