

3. The New User Source window will open. Some properties are optional, but if you're using Automatic mode, enter the following properties as appropriate.

- **Name:** DBAuth - name of the user source.
- **Failover Source:** default - failover user source ('default' is the internal user source).
- **Failover Mode:** Hard - if the source is unreachable, then use the failover source. (Can choose the Hard or Soft option).
- **Database:** MySQL - external database.
- **Mode:** Automatic - tables in the external database will be automatically created when needed.
- **Tablename Prefix:** 'auth_' is the prefix for all the tables that get created. (You can leave this field blank, but if you use a prefix when the tables get created, they will contain the specified prefix in their name).
- When finished, click **Create New User Source**.

The screenshot shows the 'User Sources' configuration interface with three stacked panels:

- Main Panel:**

Name	DBauth
Description	Database authentication
Schedule Restricted	<input type="checkbox"/> Users are only able to log in when their assigned schedule is active. (default: false)
Failover Source	default
Failover Mode	Hard
Cache Validation Timeout	60000
- Main Properties Panel:**

Database	MySQL
Mode	Automatic
- Automatic Mode Panel:**

Tablename Prefix	auth_
------------------	-------



Table Creation

The tables in the database will not be created in the database until they are needed. For example, as soon as a user or role is added, the associated tables will automatically get created.

4. Now that your Authentication profile is created, add a user. On the right, click on the **More > Manage Users** link. Click on the **Add User** link and fill in the required fields.
5. Now that your tables are created we can verify them. To view the tables, go into **Designer** and from the menu bar, select **Tools > Database Query Browser**. You will see all the tables that were created beginning with 'auth_' when the user and role get created.
6. Double click on any of the tables beginning with '**auth_**', and click **Execute**. In this example, you will see the tables associated with 'roles' and 'users' displayed in the Schema area.

The screenshot shows the Oracle Database Query Browser interface. The query editor at the top contains the SQL command: `SELECT * FROM auth_users`. Below the query is a checkbox for "Limit SELECT to: 1000 rows". The results pane displays a single row of data for a user named "test". The schema pane on the right shows the structure of the `auth_users` table, which includes columns for id (INT), fname (VARCHAR), language (VARCHAR), lname (VARCHAR), notes (VARCHAR), and passwd (VARCHAR). The entire schema pane is highlighted with a red box.

Related Topics ...

- [Internal Authentication](#)
- [Active Directory Authentication](#)
- [AD Internal Hybrid Authentication](#)
- [AD Database Hybrid Authentication](#)
- [Identity Provider Authentication Strategy](#)
- [Database Query Browser](#)

Active Directory Authentication

Active Directory User Source

The Active Directory Authentication profile uses Microsoft's Active Directory over **LDAP** (Lightweight Directory Access Protocol) to store all the users, roles, and more that make up an Authentication profile. Active Directory **Groups** are used for Ignition's **roles** and user-role mappings.

While using an Active Directory User Source, administration of users and roles is through Active Directory itself, and not manageable within Ignition. Thus adding new users to an Active Directory User Source, or modifying pre existing users, requires the modifications be made from Active Directory, usually through an AD Administrator.

Active Directory User Sources supports SASL (Simple Authentication and Security Layer). SASL is a framework for authentication and data security in Internet protocols such as LDAP.

Property Reference

Active Directory User Sources have the following properties shown in the table below, organized by category.

Note: Certain properties in the Active Directory User Source allow you to filter users, such as the **User List Filter**. These filters only determine which users will be displayed on screen. They are not authentication filters, so even if a user does not show in the list they can still authenticate and may have access to unintended areas. Be sure to configure [project security](#) appropriately to prevent this from happening!

Main Properties

Details on the Main Properties can be found on the [Classic Authentication Strategy](#) page.

Active Directory Properties

Name	Description
Domain	The Windows Domain your active Active Directory server is running on. If you aren't sure of your domain, ask your network administrator. Leave blank to set advanced properties manually.
Gateway Username	The login name for the Gateway to use when querying Active Directory. Used for retrieving the list of users and roles via LDAP.
Password	The password for the above username.
Re-type Password	Re-type password for verification.
Primary Domain Controller Host	The IP address or hostname of your primary domain controller. Example: "192.168.1.4" or "MainServer"
Primary Domain Controller Port	The port number for the primary domain controller's LDAP interface.
Secondary Domain Controller Host	The IP address or hostname of your secondary domain controller (optional). Example: "192.168.1.4" or "MainServer"
Secondary Domain Controller Port	The port number for the secondary domain controller's LDAP interface.
Use SSL	Disable to use " ldap:// " protocol, enable to use " ldaps:// "
SSO Enabled	Whether or not to use Single-Sign-On (SSO) to authenticate AD users. This gives you the ability to log into the Client or Designer automatically with the password you logged into windows with.

Note:

For Client SSO login, each project must also have the [SSO Login Project Property](#) enabled.

For Designer SSO login, [Allow Designer SSO](#) must be set in the [Gateway Settings](#).

On this page ...

- [Active Directory User Source](#)
- [Property Reference](#)
 - [Main Properties](#)
 - [Active Directory Properties](#)
 - [LDAP Search Properties](#)
 - [SASL Properties](#)
- [To Create an Active Directory User Source](#)



INDUCTIVE
UNIVERSITY

Active Directory Authentication

[Watch the Video](#)

SSO Domain	The domain that Windows users must match in order to use SSO. If blank, the main "Domain" property will be used.
------------	--

LDAP Search Properties

Name	Description
Username Prefix	This prefix will be prepended to the username before an Active Directory bind is attempted for authentication.
Username Suffix	This suffix will be appended to the username before an Active Directory bind is attempted for authentication.
Automatic Suffix	If this option is checked, and the suffix is left blank, then the suffix will automatically be assigned a value of "@<domain>".
User Search Base	<p>The base folder to search for users under, such as:</p> <p>DC=MyCompany,DC=com</p> <p>The entire subtree under this folder will be searched using the User Search Filter.</p> <p>Multiple subtrees can be specified by putting them in parenthesis, like so:</p> <p>(OU=Administrators,DC=MyCompany,DC=com)(OU=Operators,DC=MyCompany,DC=com)</p>
User Search Filter	The LDAP search filter that will be used to find a specific user. Use the placeholder {0} as a standin for the login name.
User List Filter	The LDAP search filter used when querying for the list of all users. Should restrict the type to user.
User Name Attribute	The attribute on the User object to define the username.
User Role Attribute	Attributes of this name on the User object will define the user's roles.
Role Name Attribute	The attribute of this name on the Role object will define the role's name. Leave blank to use the raw value of the attribute defined by the User Role Attribute property.
Full Name Attribute	The attribute on the User object to define the full name of the user.
Phone Attribute	The attribute name on the user object that represents the user's phone number.
Email Attribute	The attribute name on the user object that represents the user's email address.
SMS Attribute	The attribute name on the user object that represents the phone number that this user receives text messages on.
Read Timeout	The read timeout in milliseconds for LDAP operations.
Results Page Size	The number of entries returned per page of results in a query.
Role Search Base	<p>The base folder to search for roles under, such as:</p> <p>OU=Roles,DC=MyCompany,DC=com</p> <p>The entire subtree under this folder will be searched using the Role Search Filter. If you specify the root of your tree structure, the search may take a very long time.</p> <p>Multiple subtrees can be specified by putting them in parenthesis, like so:</p>

	(OU=Builtin,DC=MyCompany,DC=com)(OU=Users,DC=MyCompany,DC=com) If you leave this blank the whole subtree of the domain controller will be searched.
Role Search Filter	The LDAP search filter that will be used to locate roles.
Allow Anonymous	Determines whether the Gateway will accept blank usernames and passwords for authentication. This check takes place on the Gateway, prior to handing off any credentials to the AD server. If Security Authentication is set to None, then this property should be enabled, otherwise, blank passwords will be rejected by the Gateway. If true, authentication attempts with blank passwords will be passed through to LDAP, which may choose to accept them.
Security Protocol	Specifies the security protocol between the Gateway and AD server. The following options are available: AUTO: No security protocol is explicitly used or requested by the Gateway. SSL: SSL should be used for the connection.
Security Authentication	This property specifies how usernames and passwords are used to bind to LDAP. The following options are available: AUTO: Unspecified from the Gateway side, meaning the LDAP implementation will choose. NONE: Anonymous access. SIMPLE: Plaintext username and passwords will be used. STRONG: Usernames and passwords will be encrypted.
Referral	<p>This feature is new in Ignition version 8.1.1 Click here to check out the other new features</p> <p>Specifies how referrals are to be processed. Possible options are:</p> <p>Follow: Always automatically follow referrals. This is the default option.</p> <p>Ignore: Ignores referrals.</p> <p>Throw: Throws a ReferralException whenever a referral is encountered.</p>

SASL Properties

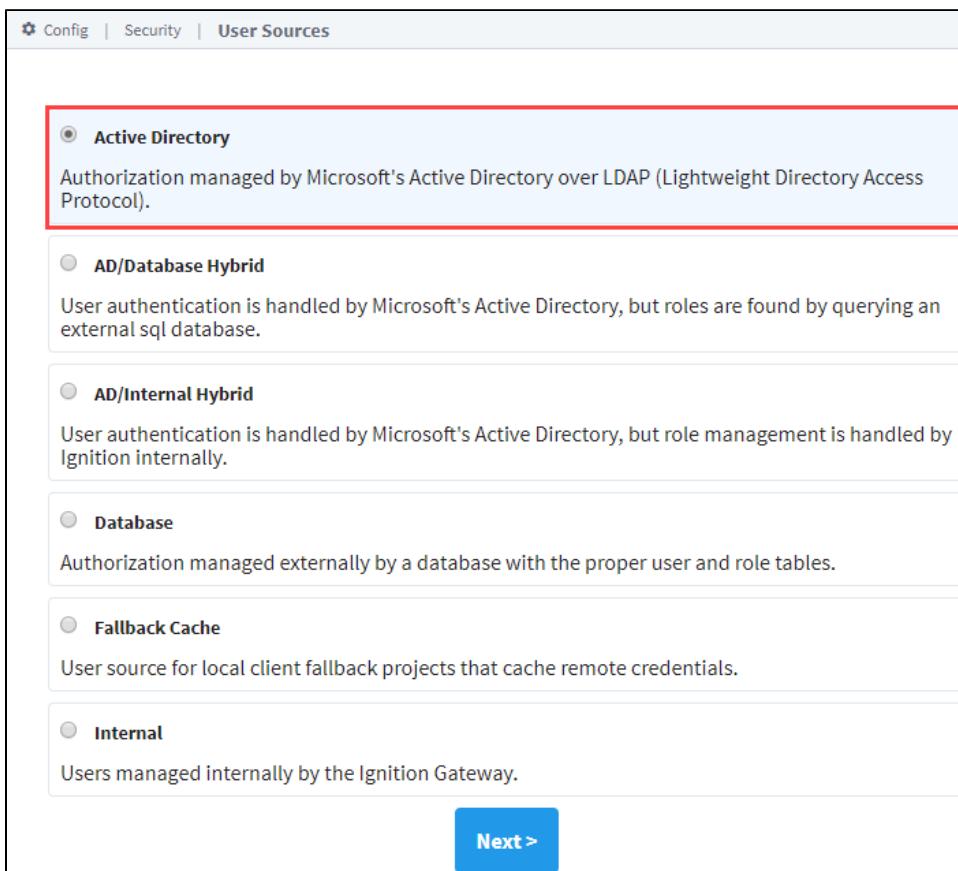
Name	Description
Mechanism	An ordered list of space-separated mechanism names. The LDAP provider will use the first mechanism for which it finds an implementation. A blank value will leave this setting unspecified. (Default is DIGEST-MD5 CRAM-MD5.)
Realm	A realm defines the namespace from which the user is selected. A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. Default is blank.
Quality of Protection	A comma-separated list of Quality-of-Protection (QoP) values, the order of which specifies the preference order. There are three well-known values: "auth" (authentication only), "auth-int" (authentication with integrity protection), and "auth-conf" (authentication with integrity and privacy protection). A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. (Default is auth-conf,auth-int,auth.)
Protection Strength	A comma-separated list of privacy protection strength values, the order of which specifies the preference order. The three possible strength values are "low", "medium", and "high". A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. Default is high,medium,low.
Mutual Authentication	Enable or disable mutual authentication. This setting will only be used by mechanisms which support it. Default is disabled.

To Create an Active Directory User Source

To configure an Active Directory User Source, you must specify the host that is acting as your primary domain controller. You can also use a secondary domain controller in case the primary is unavailable. You'll also need to specify the name of the domain and credentials for the Gateway itself to use: the Gateway needs a user account to interact with the AD server, even when it's simply querying for a list of roles.

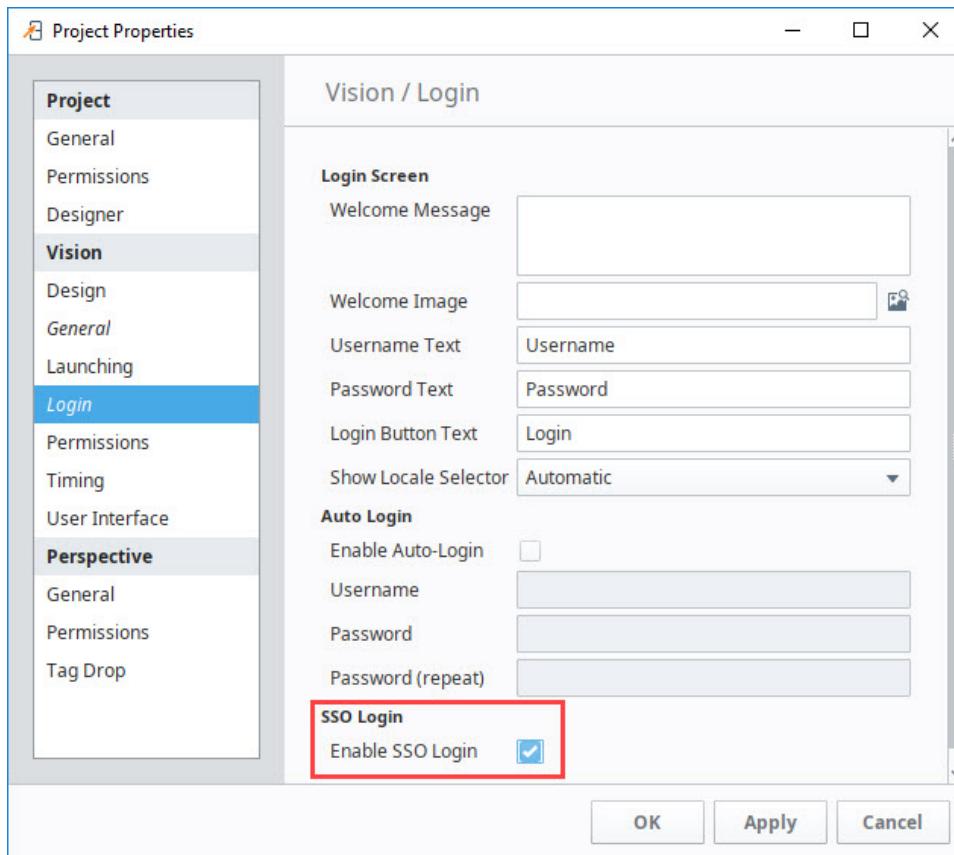
Note: When using Active Directory User Source, you may need to consult with your internal IT Department to get the required information to complete your user source setup. These settings are common to AD (not specific to Ignition), and your IT department will know what values to supply to each property.

1. On the [Gateway Webpage](#), under the Config tab, go to **Security > Users, Roles**.
The User Sources page will be displayed. Click the blue arrow, **Create new User Source**.
2. Choose the **Active Directory** authentication type, and click **Next**.



3. The New User Source window will open. Some properties are optional. In the very least, you must specify the following: **Domain**, **Gateway Username**, **Password**, **Primary Domain Controller Host**.
4. If you plan on using Single-Sign-On, then you will need to enable the **SSO Enabled** property, and continue on to the next step. In either case, click the **Create New User Source** button to create the User Source.
5. If you plan on using **Single-Sign-On**, you need to enable it per project by modifying the [project's settings](#): in the **Designer** under **Project > Properties**. Under **Vision > Login**, mark the checkbox for **Enable SSO Login**. Click **OK**.

Note: SSO is enabled on a per project basis. You have to configure it in the Active Directory authentication profile, and enable it for each project you want to use SSO with.



Related Topics ...

- Internal Authentication
- Database Authentication
- AD Internal Hybrid Authentication
- AD Database Hybrid Authentication
- Identity Provider Authentication Strategy
- Classic Authentication Strategy
- Project Security in the Designer

AD Internal Hybrid

AD/Internal User Source

The Active Directory/Internal Hybrid authentication profile type combines the Internal User Source type with the Active Directory User Source type. Active Directory is used to find all of the users, and to check their credentials when they attempt to log in. However, it allows assigning of roles, contact info, and other meta-information about a user through Ignition, then stores all this information as if it were an Internal User Source. This way, Active Directory can be consulted to see if a username/password is valid, but the management of roles does not require coordination with your IT Department, who typically controls the Active Directory system. This "best of both worlds" approach is popular for many users of Active Directory.

The AD/Internal Hybrid User Source is partially manageable in Ignition. Users cannot be added or removed, and their usernames and passwords cannot be changed. This is because this information resides in Active Directory, not within Ignition. Other information, such as user roles, contact info, schedules, are [manageable](#) in Ignition.

AD Internal Hybrid User Sources now support SASL (Simple Authentication and Security Layer). SASL is a framework for authentication and data security in Internet protocols such as LDAP.

[Gateway Settings](#)

Before you can use the User Management component to manage roles, contact info, etc., you first have to go into [Gateway Settings](#), and mark the checkbox to 'Allow User Admin.' This allows for the administration of the Gateway's system user source from the Designer and the Client. Unless this is enabled, the Vision Module's User Management component is prevented from modifying the Gateway system's user source.

Property Reference

This [User Source](#) shares many properties with the AD User Source. Please see the [Active Directory Authentication](#) page for a list of properties.

Creating an AD/Internal Hybrid User Source

To set up an AD/Internal Hybrid User Source, you must specify the host that is acting as your primary domain controller. You can also use a secondary domain controller in case the primary is unavailable. You'll also need to specify the name of the domain and credentials for the Gateway itself to use for authentication for when it queries the list of roles.

[May need to contact your internal IT Department for...](#)

When using AD/Internal Hybrid User Source, you may need to consult with your internal IT Department to get the required information to complete your user source setup.

1. On the [Gateway Webpage](#), under the Config tab, go to **Security > Users, Roles**. The User Sources page will be displayed. Click the blue arrow, **Create new User Source**.
2. Choose the **AD/Internal Hybrid** authentication type, and click **Next**.

On this page ...

- [AD/Internal User Source](#)
- [Property Reference](#)
- [Creating an AD/Internal Hybrid User Source](#)
 - [Active Directory Properties](#)
 - [SASL Properties](#)
 - [Advanced Properties](#)



INDUCTIVE
UNIVERSIT

AD Internal Hybrid

[Watch the Video](#)

Config | Security | User Sources

- Active Directory
Authorization managed by Microsoft's Active Directory over LDAP (Lightweight Directory Access Protocol).
- AD/Database Hybrid
User authentication is handled by Microsoft's Active Directory, but roles are found by querying an external sql database.
- AD/Internal Hybrid
User authentication is handled by Microsoft's Active Directory, but role management is handled by Ignition internally.
- Database
Authorization managed externally by a database with the proper user and role tables.
- Fallback Cache
User source for local client fallback projects that cache remote credentials.
- Internal
Users managed internally by the Ignition Gateway.

Next >

3. The New User Source window will open. Some properties are optional depending on how you set up your profile. Details on the Main Properties can be found on the [User Sources](#) page. Active Directory properties are listed in the table below.
4. Click **Create New User Source** to save the new user source.

Active Directory Properties

Name	Description
Domain	The Windows Domain your active Active Directory server is running on. If you aren't sure of your domain, ask your network administrator . Leave blank to set advanced properties manually.
Primary Domain Controller Host	The IP address or hostname of your primary domain controller. Example: "192.168.1.4" or "MainServer"
Primary Domain Controller Port	The port number for the primary domain controller's LDAP interface.
List Users from Active Directory	If true, Active Directory will be queried for the list of all users. If false, users must be added manually. Default is true.
Gateway Username	The login name for the Gateway to use when querying Active Directory. Used for retrieving the list of users and roles via LDAP.
Password	The password for the above username.
Password	Re-type password for verification.
SSO Enabled	Whether or not to use Single-Sign-On (SSO) to authenticate AD users. Note that projects must also have this option enabled for SSO to work. Default is false.
SSO Domain	The domain that Windows users must match in order to use SSO . If blank, the main "Domain" property will be used.

SASL Properties

Name	Description
------	-------------

Mechanism	An ordered list of space-separated mechanism names. The LDAP provider will use the first mechanism for which it finds an implementation. A blank value will leave this setting unspecified. Default is DIGEST-MD5 CRAM-MD5.
Realm	A realm defines the namespace from which the user is selected. A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. Default is blank.
Quality of Protection	A comma-separated list of Quality-of-Protection (QoP) values, the order of which specifies the preference order. There are three well-known values: "auth" (authentication only), "auth-int" (authentication with integrity protection), and "auth-conf" (authentication with integrity and privacy protection). A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. Default is auth-conf,auth-int,auth.
Protection Strength	A comma-separated list of privacy protection strength values, the order of which specifies the preference order. The three possible strength values are "low", "medium", and "high". A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. Default is high,medium,low.
Mutual Authentication	Enable or disable mutual authentication. This setting will only be used by mechanisms which support it. Default is disabled.

Advanced Properties

Name	Description
Secondary Domain Controller Host	The IP address or hostname of your secondary domain controller (optional). Example: "192.168.1.5" or "BackupServer"
Secondary Domain Controller Port	The port number for the secondary domain controller's LDAP interface. Default: 389.
Read Timeout	The read timeout in milliseconds for LDAP operations. Default is 60,000.
Results Page Size	The number of entries returned per page of results in a query. Default is 1,000.
User Listing Base	The base folder to search for users under, such as "DC=MyCompany,DC=com". The entire subtree under this folder will be searched using the User List Filter. Multiple subtrees can be specified by putting them in parenthesis, like so: "(OU=Administrators, DC=MyCompany,DC=com)(OU=Operators,DC=MyCompany,DC=com)" If you leave this blank the whole subtree of the domain controller will be searched.
User List Filter	The LDAP search filter used when querying for the list of all users. Should restrict the type to user. Default is (& (objectClass=user) (! (objectClass=computer))).
User Search Filter	The LDAP search filter to use to find a specific user. Use the placeholder {0} as a standin for the login name. Default is (& (objectClass=user) (sAMAccountName={0})).
Username Attribute	The attribute on the User object to define the username. Default is sAMAccountName.
Username Prefix	This prefix will be prepended to the username before an Active Directory bind is attempted for authentication.
Username Suffix	This suffix will be appended to the username before an Active Directory bind is attempted for authentication.
Automatic Suffix	If this option is checked, and the suffix is left blank, then the suffix will automatically be assigned a value of "@<domain>". Default is true.
Allow Anonymous	If true, authentication attempts with blank passwords will be passed through to LDAP, which may choose to accept them. Default is false.
Security Protocol	Auto or SSL. Default is Auto.
Security Authentication	Auto, None, Simple, or Strong. Default is Auto.

Referral

This feature is new in Ignition version **8.1.1**.
[Click here](#) to check out the other new features

Specifies how referrals are to be processed. Possible options are:

Follow: Always automatically follow referrals. This is the default option.

Ignore: Ignores referrals.

Throw: Throws a ReferralException whenever a referral is encountered.

Related Topics ...

- [Internal Authentication](#)
- [Database Authentication](#)
- [Active Directory Authentication](#)
- [AD Database Hybrid Authentication](#)
- [Identity Provider Authentication Strategy](#)
- [User Management Component](#)

AD Database Hybrid

AD/Database User Source

This AD/Database Hybrid User Source is not [manageable](#) from within Ignition. Users/passwords must be administered through [Active Directory](#), and roles, contact info, and so on, must be administered directly through the database. The way AD/Database Hybrid works, is it has all the same information requirements as the other authentication profiles, but it also has a number of Database properties. You need to specify a database (i.e., MySQL) to store information, and set up queries that you want to use. You must also specify the host that is acting as your primary domain controller, and a secondary domain controller in case the primary is unavailable. You'll also need to specify the name of the domain and credentials for the Gateway itself to use for authentication when it queries the list of roles.

AD/Database User Sources now support SASL (Simple Authentication and Security Layer). SASL is a framework for authentication and data security in Internet protocols such as LDAP.

Property Reference

This [User Source](#) shares many properties with both the AD User Source and Database User Source.

- See the [Active Directory Authentication](#) page for a list of Active Directory User Source related properties.
- See the [Database Authentication](#) page for a list of Database User Source related properties.

On this page ...

- [AD/Database User Source](#)
- [Property Reference](#)
- [Creating an AD/Database Hybrid User Source](#)
 - [Active Directory Properties](#)
 - [Database Properties](#)
 - [Advanced Properties](#)
 - [SASL Properties](#)



INDUCTIVE
UNIVERSITY

AD Database Hybrid

[Watch the Video](#)

Creating an AD/Database Hybrid User Source

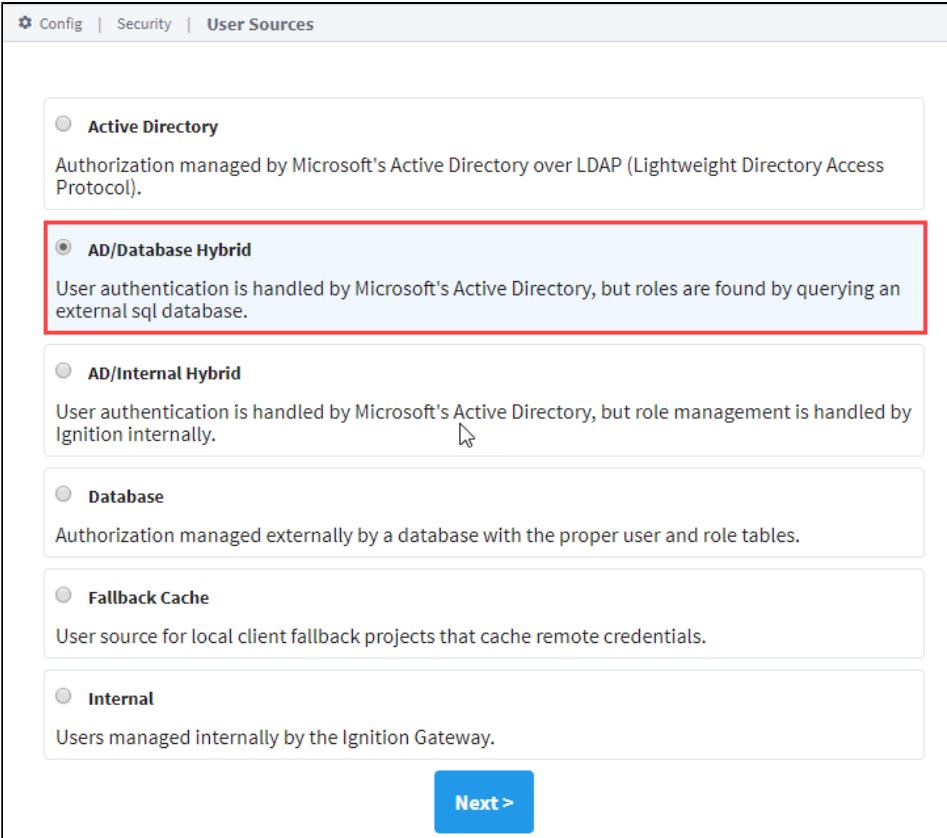
1. On the [Gateway Webpage](#), under the Config tab, go to [Security > Users, Roles](#).
The [Classic Authentication Strategy](#) page will be displayed. Click the blue arrow, [Create new User Source](#).
2. Choose the **AD/Database Hybrid** authentication type, and click **Next**.

Note: When using AD/Database Hybrid User Source, you may need to consult with your internal IT Department to get the required information to complete your user source setup.

3. The New User Source window will open. Some properties are optional depending on how you set up your profile. Details on the Main Properties can be found on the [User Sources](#) page. The Active Directory Properties and Database properties are listed in the tables below.
4. Click **Create New User Source** to save the new user source.

Active Directory Properties

Name	Description
Domain	The Windows Domain your active Active Directory server is running on. If you aren't sure of your domain, ask your network administrator . Leave blank to set advanced properties manually.
Primary Domain Controller Host	The IP address or hostname of your primary domain controller. Example: "192.168.1.4" or "MainServer"
Primary Domain Controller Port	The port number for the primary domain controller's LDAP interface.
List	If true, Active Directory will be queried for the list of all users. If false, users must be added manually. (Default is true.)

Users from Active Directory	
Gateway Username	The login name for the Gateway to use when querying Active Directory. Used for retrieving the list of users and roles via LDAP.
Password	The password for the above username.
Password	 <p>The screenshot shows the 'User Sources' configuration page. At the top, there are three tabs: 'Config', 'Security', and 'User Sources'. The 'User Sources' tab is active. Below the tabs, there is a list of authentication methods:</p> <ul style="list-style-type: none"> Active Directory: Authorization managed by Microsoft's Active Directory over LDAP (Lightweight Directory Access Protocol). AD/Database Hybrid: This option is highlighted with a red box. User authentication is handled by Microsoft's Active Directory, but roles are found by querying an external sql database. AD/Internal Hybrid: User authentication is handled by Microsoft's Active Directory, but role management is handled by Ignition internally. Database: Authorization managed externally by a database with the proper user and role tables. Fallback Cache: User source for local client fallback projects that cache remote credentials. Internal: Users managed internally by the Ignition Gateway. <p>At the bottom right of the screenshot is a blue 'Next >' button.</p> <p>Re-type password for verification</p>
SSO Enabled	Whether or not to use Single-Sign-On (SSO) to authenticate AD users. Note that projects must also have this option enabled for SSO to work.
SSO Domain	The domain that Windows users must match in order to use SSO . If blank, the main "Domain" property will be used.

Database Properties

Name	Description
Database	Dropdown list. Choose the database connection this authentication profile will use.
User Properties Query	A query that returns the basic properties for a single user. Supported return columns are [username, firstname, lastname, schedule, language, notes].
Role List Query	A query that returns all possible roles that any user could have. The role names must be returned in the first column of the query's results
User's Roles Query	A query that returns all of the roles that the provided user belongs to. The roles must be strings (i.e., the role names), and must be in the first column of the query's results. The username will be inserted into this query as a parameter.
Contact Info Query	A query that returns all of the contact info for the user. The first column must be the contact type, the second column the contact value, and the third column the name of a schedule. Optional, may be blank.
Schedule Adjustment Query	A query that returns the upcoming schedule adjustments for the user. Columns must be Start(date), End(date), Available(boolean), Note(string). Optional, may be blank.

Extra Properties Query	A query that returns name, value pairs of extra properties for the user. Will be run with one parameter: the username. Optional, may be blank.
List Users Query	A query that returns a row containing each username. Only used if "List Users from Active Directory" is false. There must be at least one column: the username. Other columns are optional, supported columns are: [username, firstname, lastname, schedule, language, notes].

Advanced Properties

Name	Description
Secondary Domain Controller Host	The IP address or hostname of your secondary domain controller (optional). Example: "192.168.1.5" or "BackupServer"
Secondary Domain Controller Port	The port number for the secondary domain controller's LDAP interface. Default: 389.
Read Timeout	The read timeout in milliseconds for LDAP operations. Default is 60,000.
Results Page Size	The number of entries returned per page of results in a query. Default is 1,000.
User Listing Base	The base folder to search for users under, such as "DC=MyCompany,DC=com". The entire subtree under this folder will be searched using the User List Filter. Multiple subtrees can be specified by putting them in parenthesis, like so: "(OU=Administrators, DC=MyCompany,DC=com)(OU=Operators,DC=MyCompany,DC=com)" If you leave this blank the whole subtree of the domain controller will be searched.
User List Filter	The LDAP search filter used when querying for the list of all users. Should restrict the type to user. Default is (& (objectClass=user) (! (objectClass=computer))).
User Search Filter	The LDAP search filter to use to find a specific user. Use the placeholder {0} as a standin for the login name. Default is (& (objectClass=user) (sAMAccountName={0})).
Username Attribute	The attribute on the User object to define the username. Default is sAMAccountName.
Username Prefix	This prefix will be prepended to the username before an Active Directory bind is attempted for authentication.
Username Suffix	This suffix will be appended to the username before an Active Directory bind is attempted for authentication.
Automatic Suffix	If this option is checked, and the suffix is left blank, then the suffix will automatically be assigned a value of "@<domain>". Default is true.
Allow Anonymous	If true, authentication attempts with blank passwords will be passed through to LDAP, which may choose to accept them. Default is false.
Security Protocol	Auto or SSL. Default is Auto.
Security Authentication	Auto, None, Simple, or Strong. Default is Auto.
Referral	<p>This feature is new in Ignition version 8.1.1 Click here to check out the other new features</p> <p>Specifies how referrals are to be processed. Possible options are:</p> <p>Follow: Always automatically follow referrals. This is the default option.</p> <p>Ignore: Ignores referrals.</p>

Throw: Throws a ReferralException whenever a referral is encountered.

SASL Properties

Name	Description
Mechanism	An ordered list of space-separated mechanism names. The LDAP provider will use the first mechanism for which it finds an implementation. A blank value will leave this setting unspecified. (Default is DIGEST-MD5 CRAM-MD5.)
Realm	A realm defines the namespace from which the user is selected. A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. (Default is blank.)
Quality of Protection	A comma-separated list of Quality-of-Protection (QoP) values, the order of which specifies the preference order. There are three well-known values: "auth" (authentication only), "auth-int" (authentication with integrity protection), and "auth-conf" (authentication with integrity and privacy protection). A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. (Default is auth-conf,auth-int,auth.)
Protection Strength	A comma-separated list of privacy protection strength values, the order of which specifies the preference order. The three possible strength values are "low", "medium", and "high". A blank value will leave this setting unspecified. This setting will only be used by mechanisms which support it. (Default is high,medium,low.)
Mutual Authentication	Enable or disable mutual authentication. This setting will only be used by mechanisms which support it. (Default is disabled.)

Related Topics ...

- [Internal Authentication](#)
- [Database Authentication](#)
- [Active Directory Authentication](#)
- [AD Internal Hybrid Authentication](#)
- [Identity Provider Authentication Strategy](#)
- [Verify a User on a User Source](#)
- [Classic Authentication Strategy](#)
- [Database Query Browser](#)

Fallback Cache Authentication

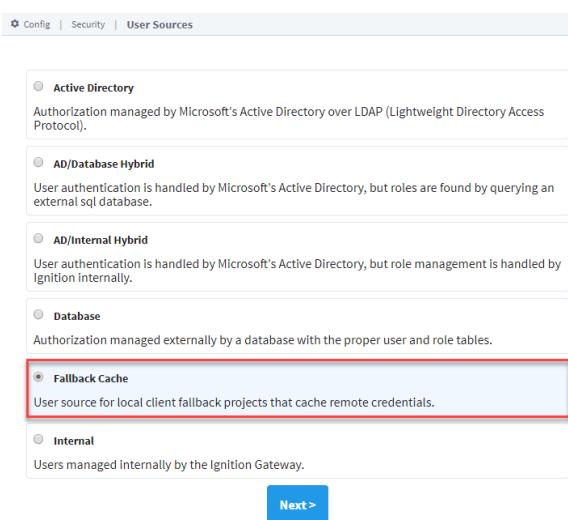
This User Source was developed specifically for a system that is using [Local Vision Client Fallback](#), and allows you to cache the login credentials from a remote user source. This means your users can still log in with their normal username/password on a Local Vision Client Fallback project, even when the network connection is unavailable.

Note: Fallback Cache Authentication does not work with [Identity Provider Authentication Strategy](#). This User Source will only function with Vision clients and user sources using [Classic Authentication Strategy](#)

Creating the Fallback Cache User Source

The Fallback Cache User Source is created in a similar fashion to any other User Source:

1. On the [Gateway Webpage](#), go to the **Config** tab. Select **Security > User, Roles**.
2. Click on the [Create new User Source...](#) link.
3. Select the Fallback Cache option and click the **Next** button.



4. Type in a name for the new User Source and click [Create New User Source](#).
5. Details on the Main Properties can be found on the [User Sources](#) page. You can also set Cache Retention as follows.

Fallback Cache Properties	
Cache Retention	Number of days that the cache will retain recently used credentials. This property determines the number of days credentials will be stored in the cache. Once exceeded, the credentials will be removed from the cache. (Default: 15)

6. Click the [Create New User Source](#) button.

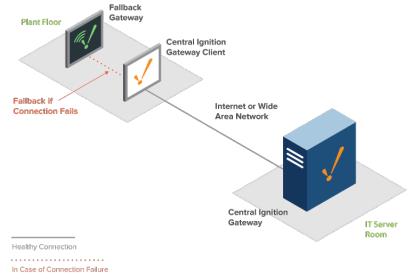
Populating the Cache

Users and Roles can not be manually added to the Fallback Cache. Instead, they are automatically copied from remote Gateways. This type of User Source is normally configured on an [Edge Gateway](#), but can be utilized on an Ignition Gateway.

- Before the Fallback Cache will populate, both the **Central Ignition Gateway** and **Fallback Gateway** must be connected over the [Gateway Network](#). The credentials are passed from a User Source on the **Central Ignition Gateway** to the Fallback Cache over the Gateway Network.
- A Fallback Cache User Source must exist on the **Fallback Gateway**.
- A client must be launched on the **Fallback Gateway** from the **Central Ignition Gateway**. If the user successfully authenticates against the **Central Ignition Gateway's** User Source, then the credentials are cached into the Fallback Cache.

On this page ...

- [Creating the Fallback Cache User Source](#)
- [Populating the Cache](#)



Related Topics ...

- [User Source](#)

Verify a User on a User Source

You can verify that a user exists in a given [User Source](#), if the password is correct, what roles a user has, and any other information added about the user.

Verify a User

1. On the Gateway Webpage, go to the **Config** tab.
2. Choose **Security > Users, Roles** from the menu on the left.
The **User Sources** page is displayed.
3. Select the **Verify an Authentication Profile** link.

The screenshot shows a table with columns for Name, Type, and Description. There are two rows:

- default**: Internal. This is the default and always present internal authentication profile. Buttons for 'manage users' and 'edit' are shown.
- opcua-module**: Internal. OPC-UA clients will authenticate against this profile by default. Buttons for 'More' and 'edit' are shown.

A red box highlights the link [Verify an Authentication Profile...](#).

4. The Verify Authentication Profile window will appear. Choose a **Profile** from the dropdown list. Enter a **Username** and **Password**. Click **Test Login**. Ignition will test the credentials then display the results of the validity test.

The screenshot shows a 'Test Login' form with the following fields:

- Profile:** default
- Username:** admin
- Password:** (masked)

A blue 'Test Login' button is at the bottom.

If the Login is incorrect, an error message will appear stating that the Login failed for a specified user.

If the Login is correct, a successful message will appear for a specified user along with their information.

The screenshot shows a 'Login Successful' window with the following content:

- Success Message:** ✓ Login succeeded for user "admin"
- User's roles:**
 - Username: admin
 - First Name: Admin
 - Last Name:
 - Schedule: Always
 - Operator
 - Administrator
 - Contact Info
 - email / admin@inductiveautomation.com
 - phone / 800-266-7798
 - Extended Properties
- Test again...** button at the bottom.

On this page ...

- [Verify a User](#)



Verifying an Authentication Profile

[Watch the Video](#)

Identity Provider Authentication Strategy

An Identity Provider (IdP) offers a way for users to log in to Ignition using credentials stored outside of Ignition. An IdP creates, maintains, and manages identity (login) information while providing authentication services to Ignition. This provides a secure login that allows Ignition to use SSL and two-factor authentication (2FA).

Identity Providers (IdPs) offer user authentication as a service. An IdP creates, maintains, and manages identity information for principals while providing authentication services to relying party applications within a federation or distributed network. Authentication of the user is handled by the IdP. Ignition can connect to these three different types of IdPs:

- Ignition's internal IdP
- OpenID Connect 1.0
- Security Assertion Markup Language (SAML)

Your organization's IT may have some sort of existing integration with an Identity Provider. Some popular Identity Providers are listed below.

- Ping Identity
- Okta
- Active Directory Federation Services
- Duo

IdPs are set up at the Gateway level. **Security Levels** are also set through the Gateway. The Security Levels enable you to define a hierarchy of access inside a Perspective Session.

This feature is new in Ignition version **8.1.0**.
[Click here](#) to check out the other new features

As of release 8.1, Identity Providers can also be used in the Vision module, the Designer, and on the Gateway. The Identity Provider strategy redirects the user to their IdP in their web browser in order to authenticate. The System Identity Provider setting controls which Identity Provider the user is redirected to.

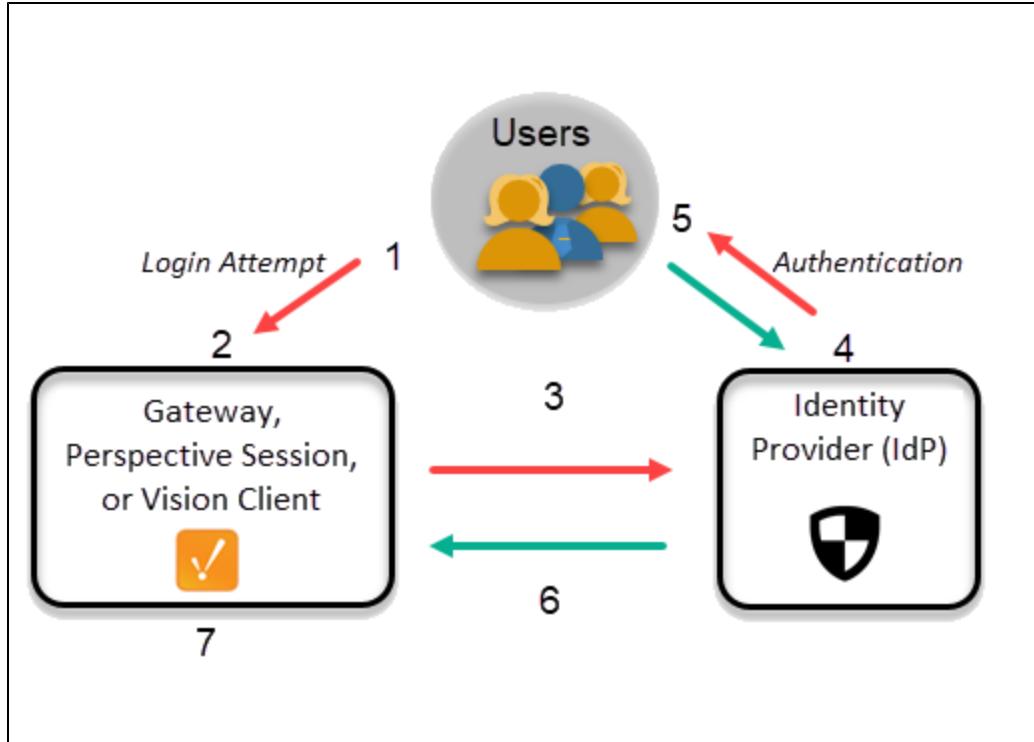
Note: If your browser is not supported, you will get an error message.

On this page ...

- [Identity Provider Authentication Workflow](#)
- [Using Identity Providers](#)

Identity Provider Authentication Workflow

The following diagram illustrates how IdP authentication works.



Using Identity Providers

The first step in using Identity Providers is to configure them. For the steps for configuring Internal Ignition IdP, OpenID Connect 1.0, or Security Assertion Markup Language (SAML), go to [Configuring Identity Providers](#).

Once an Identity Provider has been configured, there are a few things that can be done to test and adjust how it works. You can [map the attributes](#) that are returned in the IdP response document to more familiar user properties that are available to use within the project. You can add [rules to custom security levels](#) that determine when a user falls into the level. Overrides can be given to users in the form of [User Grants](#), so that they are granted certain security levels regardless of the rules. Finally, you can [test out the IdP](#) by logging in with a user to confirm what is returned in the response document.

The screenshot shows the Ignition Config interface with the following navigation path: Home > Security > Identity Providers.

The left sidebar is organized into sections:

- SYSTEM**: Overview, Backup/Restore, Ignition Exchange, Licensing, Modules, Projects, Redundancy, Gateway Settings.
- NETWORKING**: Web Server, Gateway Network, Email Settings.
- SECURITY**: General, Auditing, Users, Roles, Service Security, **Identity Providers** (which is currently selected), and Security Levels.

The main content area displays the "Identity Providers" page with the following details:

- Header: Filter (type to filter), View (20), Page (1 of 1).
- Table:

Name	Type	Description	Action
default	Ignition	Automatically generated Ignition Identity Provider which uses the User Source Profile named "default".	More ▾ Settings
Okta_2018	OpenID Connect 1.0		More ▾ Settings
- Buttons: Create new Identity Provider..., Import Identity Provider... (both in blue).
- Pagination: Page (1 of 1).

In This Section ...

Configuring Identity Providers

Registering the Ignition Gateway

Before configuring an Identity Provider on the Ignition Gateway, it must first be registered as an Identity Provider Client. Your Identity Provider will have a workflow to register, and it will most likely request something called a **return URL** or **redirect URI**. The paths provided utilize your Gateway's address /hostname, and they change depending on the type of provider.

Note: The same redirect URI is used for login and logout.

OpenID Connect Providers (OP):

OpenID Connect Providers

`http://yourGatewayAddress:Port/data/federate/callback/oidc`

SAML Providers:

SAML Providers

`http://yourGatewayAddress:Port/data/federate/callback/saml`

On this page ...

- Registering the Ignition Gateway
- Secure Integration with IdPs
- Configure an Identity Provider
- Common Properties
- Ignition Identity Provider
 - Badge Settings
 - Built-In Attributes
 - Remember Me Example
- OpenID Connect Providers
 - Importing Metadata from the Provider
 - Configuring the Provider
 - JSON Web Key Configuration
- Security Assertion Markup Language (SAML) Providers
 - Importing Metadata from the Provider
 - Configuring the Provider
 - SAML Signature Verifying Key Configuration
- IdP Examples and Troubleshooting

Secure Integration with IdPs

You should always use the secure versions of those redirect URIs (`https`) in production environments. To do this you must [enable SSL/TLS](#) in Ignition and install a valid certificate. This is the best practice for maintaining a secure integration with third party Identity Providers.



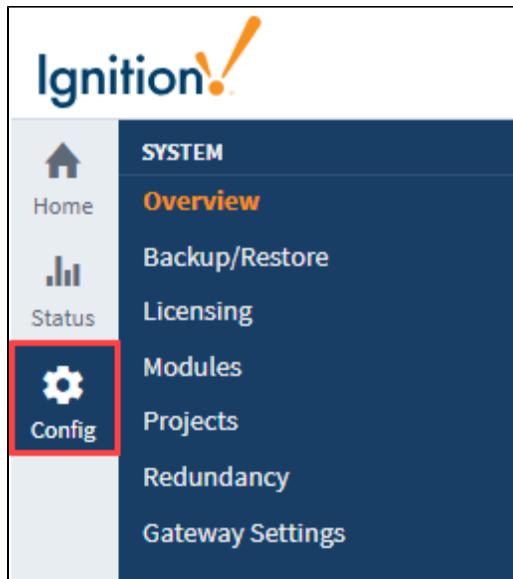
Configuring Identity Providers

[Watch the Video](#)

Configure an Identity Provider

Although there are several types, the general workflow for creating an Identity Provider is the same.

1. On the Gateway Webpage, click on the **Config** tab.



- Under the Security section, click on **Identity Providers**. The Identity Providers screen is displayed. This screen will list all IdPs that have been configured. You can change filter by name or adjust the number of IdPs displayed in the view.

The screenshot shows the 'Identity Providers' screen. The left sidebar includes 'Identity Providers' which is also highlighted with a red box. The main area displays a table with two rows:

Name	Type	Description	Action
Default_User_Source	Ignition		More Delete
Okta_2018	OpenID Connect 1.0		More Delete

Below the table are two buttons: 'Create new Identity Provider...' and 'Import Identity Provider...', both of which are highlighted with red boxes.

- Click on **Create New Identity Provider...**
- Choose the type of provider. The current options are **Ignition**, **OpenID Connect 1.0** or **Security Assertion Markup Language 2.0 (SAML)**.

The screenshot shows the 'Create New' screen for identity providers. The left sidebar has a 'Config' icon highlighted with a red box. The main area shows three options:

- Ignition**: Described as useful in cases where no external identity provider is available.
- OpenID Connect 1.0**: Described as an authentication layer on top of OAuth 2.0.
- Security Assertion Markup Language 2.0**: Described as an open standard for exchanging authentication and authorization data between parties.

At the bottom are 'Cancel', 'Step 1 of 2', and 'Next' buttons, with 'Next' being highlighted with a red box.

- Click the **Next** button.

- Configure the adapter. This step varies based on the type of provider. Please see the reference tables below for a description of properties.
- Once you've filled in the properties, click **Save**.

Common Properties

All Identity Provider types share the following properties:

Property Name	Description	Required?
Provider Name	<p>The name of the adapter. Adapter names must be unique, so no two adapters on the same Gateway may have the same name.</p> <p>The naming conventions for IdPs are as follows:</p> <ul style="list-style-type: none"> IdP names must begin with an underscore or alpha character. The remaining characters in the name must be either underscores or alphanumeric. IdP names are not case sensitive. 	Yes
Provider Description	A description of the provider.	No
Provider Type	The type of Identity Provider. The value for this field comes from the previous screen. It cannot be changed here.	Yes

Ignition Identity Provider

The Ignition Identity Provider has the following properties:

Property Name	Description	Required?
User Source	The User Source for this IdP. In order to properly authenticate users, the Ignition Identity Provider must be able to query the list of users from the underlying user source profile.	Yes
Session Inactivity Timeout	<p>This feature is new in Ignition version 8.1.0 Click here to check out the other new features</p> <p>The number of minutes which must elapse before expiring a session due to user inactivity. Sessions will not timeout if set to any number less than or equal to zero.</p>	Yes
Session Expiration	<p>This feature is new in Ignition version 8.1.0 Click here to check out the other new features</p> <p>The maximum number of minutes a session may exist before it is expired. Sessions will not have a max lifetime if set to any number less than or equal to zero.</p>	Yes
Remember Me Expiration	<p>This feature is new in Ignition version 8.1.0 Click here to check out the other new features</p> <p>The maximum number of hours a user will be remembered if they elect to be remembered. Remember Me is disabled when this value is set to any number less than or equal to zero. For more information on this option, see the Remember Me section below.</p>	Yes
Authentication Methods	You can opt into Badge based authentication for the IdP by enabling the "Badge" Authentication Method. The "Default" radio button determines which option users first see when attempting to authenticate against the IdP.	Yes

Name	Enabled	Default
Username and Password	<input checked="" type="checkbox"/>	<input checked="" type="radio"/>
Badge	<input type="checkbox"/>	<input type="radio"/>

Badge Secret	Choose whether or not the user is required to enter a secret (password) along with their badge scan. Additional option can be checked to require the user to enter their password in addition to scanning their badge.	Yes
--------------	--	-----

Badge Settings

Property Name	Description	Required?
Badge Secret	Choose whether or not the user is required to enter a secret (password) along with their badge scan. Check to require the user to enter their password in addition to scanning their badge. Default is false (not checked).	Yes

Built-In Attributes

The following attributes are available in the Ignition IdP.

Attribute	Type	Description	Example
auth_time	Date	Represents the time the user last authenticated.	<pre>// Check if it has been within 15 minutes since the last // authentication attempt dateDiff ({idp-attributes: auth_time}, now(), "minutes") <= 15</pre>
challenged	Boolean	<p>Signifies if the user provided credentials at the last login.</p> <p>If true, then the user was asked to re-validate their credentials the last time they attempted to login.</p> <p>If false, then they were not challenged to re-validate their credentials during the last login attempt. This can happen when a login request was made after a user was already authenticated. For example, if a user was already authenticated in a Perspective Session, and a separate call to <code>system.perspective.login</code> function was made with the forceAuth parameter set to false, meaning the user did not provide credentials during the last authentication challenge.</p>	<pre>// Returns True or False, depending on whether or not the user // provided credentials at the last login. {idp-attributes: challenged}</pre>

Remember Me Example

This feature is new in Ignition version **8.1.0**
[Click here](#) to check out the other new features

The Remember Me option allows your login to be remembered for a set amount of time, even if you close your browser or restart your Gateway. When set, you will be remembered on this device for the specified number of hours without needed to log in again.

Note: Note that updating the Ignition Gateway to a new version will cause the device to "forget" the remembered user for some update versions, namely 8.1.2.

Caution: This option is not recommended if you are using a public or shared device.

To set up Remember me, do the steps that follow:

1. On the Gateway Webpage, click on the **Config** tab. Scroll down to **Security > Identity Providers**.
2. For the Ignition Identity Provider you'd like to configure, click on the **More** option and choose **Settings**.

The screenshot shows the 'Identity Providers' configuration page. It lists one provider named 'default' of type 'Ignition'. The 'Action' column for this provider has a 'More' button highlighted with a red box. Below the table, there are links to 'Create new Identity Provider...' and 'Import Identity Provider...', and a 'Settings' link which is also highlighted with a red box. A dropdown menu for 'Settings' is open, showing options: 'User Attribute Mapping' and 'User Grants'.

3. On the Settings page, scroll down to the **Provider Configuration** section.
4. For the Remember Me Expiration option, enter a value greater than zero. For this example, we set the option to **two** hours.

The screenshot shows the 'default - Settings' page under 'Provider Configuration'. It includes fields for 'User Source *' (set to 'default'), 'Session Inactivity Timeout *' (set to '30'), 'Session Expiration *' (set to '0'), and 'Remember Me Expiration *' (set to '2'). The 'Remember Me Expiration' field is highlighted with a red box. Below these is a section for 'Authentication Methods *' with a table:

Name	Enabled	Default
Username and Password	<input checked="" type="checkbox"/>	<input checked="" type="radio"/>
Badge	<input type="checkbox"/>	<input type="radio"/>

5. Click Save to save your changes.

To enable Remember Me for your login, do a test login:

1. On the Gateway Webpage, click on the **Config** tab. Scroll down to **Security > Identity Providers**.
2. For the Ignition Identity Provider you'd like to configure, click on the **More** option and choose **Test Login**.
3. Enter your password and select the Remember Me option.

Log In to continue

Log in as:
admin
Password
••••••••

Remember me on this device ([what's this?](#))

CONTINUE

4. Click the **Continue** button.
5. Your login will now be remembered for the amount of hours that were specified in the Gateway setting (in this example, it is 2 hours).

OpenID Connect Providers

OpenID Connect Providers (OP) properties are listed in the following tables. The values on many of these properties may require you to refer to information from your third-party IdP.

Importing Metadata from the Provider

This method is preferred because of its ease-of-use and accuracy. After importing, you will only need to add your client ID and secret manually. (However you can revise the imported data if needed as well.)

Property Name	Description
Import from URL	URL to the OpenID Provider Configuration document. Typically, if the issuer is "https://example.org/foo" then the metadata URL would be "https://example.org/foo/.well-known/openid-configuration"
Import From File	File must be a JSON document with the properties described in section 3 (OpenID Provider Metadata) of the OpenID Connect Discovery 1.0 specification.

Configuring the Provider

Most OpenID Providers will require registering Ignition as a client. After the registration process is complete, the provider will generate a client ID and secret for Ignition, which is required below. This gives Ignition the ability to communicate securely with the provider. Most providers will also require a set of redirect URIs. The redirect URI for this Ignition Gateway is: <http://docker.ia.local:51276/data/federate/callback/oidc>

Property Name	Description	Required?
Client ID	The client identifier registered within the identity provider. This value is provided by the Identity Provider.	Yes
Client Secret	The client secret registered within the identity provider. This value is provided by the Identity Provider.	Yes
Authorization URL	URL of the OP's OAuth 2.0 Authorization Endpoint.	Yes
Token URL	URL of the OP's OAuth 2.0 Token Endpoint.	Yes

Logout URL	Optional URL at the OP to which an RP can perform a redirect to request that the end user be logged out at the OP.	No
JSON Web Keys URL	URL of the OP's JSON Web Key Set document.	No
Use Json Web Keys URI	If checked, then identity provider public keys will be automatically downloaded from given JSON Web Keys URL. New keys will be automatically fetched when the identity provider generates new keys. If unchecked, then the static set of JSON Web Keys (configured below) are used, so when the identity provider rotates keys, they must be manually added to this configuration.	No
User Info URL	Optional URL to retrieve UserInfo claims from the provider. Resulting claims are typically determined by the scopes listed under the Scope setting.	No
Issuer	Entity that issues a set of claims.	Yes
Supported ID Token Signing Algorithm Values	A list of the JSON Web Signature (JWS) signing algorithms supported by the OP for the ID Token to encode the claims in a JWT.	Yes
Scope	A list of scopes which will be sent for each auth request to the OP. Commonly used scopes would be email and profile but check your Identity Provider's documentation for more information.	No
JSON Web Key Config	A list of signing key(s) the RP uses to validate signatures from the OP.	No

JSON Web Key Configuration

Property Name	Description	Required?
Key Type	The cryptographic algorithm family used with the key. Options are EC, RSA or oct.	Yes
Public Key Use	The intended use of the public key. Options are sig or eng.	No
Key Operations	The operation(s) for which the key is intended to be used.	No
Algorithm	The algorithm intended for use with the key.	Yes
Key ID	Used to match a specific key.	No
X.509 URL	A URI that refers to a resource for an X.509 public key certificate or certificate chain. The identified resource MUST provide a representation of the certificate or certificate chain that conforms to RFC 5280 in PEM-encoded form, with each certificate delimited as specified in Section 6.1 of RFC 4945.	No
X.509 Certificate Chain	The "x5c" (X.509 certificate chain) parameter contains a chain of one or more PKIX certificates. Each entry must be a base64-encoded (Section 4 of RFC4648 -- not base64url-encoded) DER PKIX certificate value.	No
X.509 Certificate SHA-1 Thumbprint	A base64url-encoded SHA-1 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate.	No
X.509 Certificate SHA-256 Thumbprint	A base64url-encoded SHA-256 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate.	No

There are some additional properties, that depend on which Key Type is selected.

Key Type: EC

Property Name	Description	Required?
crv (Curve)	The cryptographic curve used with the key.	Yes
x (X Coordinate)	The x coordinate for the Elliptic Curve point represented as the base64url encoding of the octet string representation of the coordinate.	Yes
y (Y Coordinate)	The y coordinate for the Elliptic Curve point represented as the base64url encoding of the octet string	No

	representation of the coordinate.	
d (ECC Private Key)	The Elliptic Curve private key value represented as the base64url encoding of the octet string representation of the private key value.	No

Key Type: RSA

Property Name	Description	Required?
n (Modulus)	The modulus value for the RSA public key represented as a Base64urlUInt-encoded value.	Yes
e (Exponent)	The exponent value for the RSA public key represented as a Base64urlUInt-encoded value.	Yes
d (Private Exponent)	The private exponent value for the RSA public key represented as a Base64urlUInt-encoded value.	No
p (First Prime Factor)	The first prime factor represented as a Base64urlUInt-encoded value.	No
q (Second Prime Factor)	The second prime factor represented as a Base64urlUInt-encoded value.	No
dp (First Factor CRT Exponent)	The Chinese Remainder Theorem (CRT) exponent of the first factor represented as a Base64urlUInt-encoded value.	No
dq (Second Factor CRT Exponent)	The CRT exponent of the second factor represented as a Base64urlUInt-encoded value.	No
qi (First CRT Coefficient)	The CRT coefficient of the second factor represented as a Base64urlUInt-encoded value.	No
oth (Other Primes Info)	Information about any third and subsequent primes, should they exist. Each new Prime added will provide users with new Prime Factor, Factor CRT Exponent, and Factor CRT Coefficient properties, all of which are required.	No

Key Type: oct

Property Name	Description	Required?
k (Key Value)	The value of the symmetric (or other single-values) key represented as the base64url encoding of the octet sequence containing the key value.	Yes

Security Assertion Markup Language (SAML) Providers

The properties for Security Assertion Markup Language (SAML) are listed in the following tables. The values on many of these properties may require you to refer to information from your third-party IdP.

Importing Metadata from the Provider

This method is preferred because of its ease-of-use and accuracy. After importing, you will only need to add your client ID and secret manually. (However you can revise the imported data if needed as well.)

Property Name	Description
Import from URL	URL to the SAML Identity Provider Metadata document.
Import From File	File must be an XML document which conforms to the SAML 2.0 metadata schema described in saml-metadata-2.0-os.

The SAML Service Provider (SP) metadata for an Ignition Gateway may be accessed at the following URL: <http://<ipaddress>:<port>/data/saml/metadata/sp>.

The Assertion Consumer Service (ACS) URL for this Ignition Gateway is: <http://<ipaddress>:<port>/data/federate/callback/saml>

Both of these addresses assume you know the IP Address and port of your Ignition install. For example, if you are on the computer Ignition is installed on, you could use: <http://localhost:8088/data/saml/metadata/sp> for the SP metadata.

This feature is new in Ignition version 8.1.1
[Click here](#) to check out the other new features

SAML IdPs may send the Base64-encoded SAML Response in a line-wrapped form (with new line characters such as `\r` and `\n` separating each line). As of release 8.1.1, Ignition's 2.0 SP implementation can handle both line-wrapped and non-line-wrappedBase64-encoded SAML responses.

Configuring the Provider

Property Name	Description	Required?
Entity ID	The Identity Provider's Entity ID.	Yes
Assertion Consumer Service (ACS) Binding	The expected binding used by the Identity Provider when interacting with Ignition's Assertion Consumer Service.	Yes
Name ID Format	The expected name ID format for subjects of assertions resulting from Authn Requests. Options are UNSPECIFIED, EMAIL_ADDRESS, X509 SUBJECT_NAME, WINDOWS_DOMAIN_QUALIFIED_NAME, KERBEROS_PRINCIPAL_NAME, ENTITY_IDENTIFIER, PERSISTENT_IDENTIFIER, TRANSIENT_IDENTIFIER.	Yes
Single Sign-On (SSO) Service URL	The Identity Provider's Single Sign-On (SSO) Service URL.	Yes
Single Sign-On (SSO) Service Binding *	The binding Ignition will use for sending Authn Requests to the Identity Provider's Single Sign-On (SSO) Service.	Yes
Force Authn	Check this box to force complying Identity Providers to authenticate the user each time instead of relying on a previous security context. See section 3.4.1 of saml-core-2.0-os for more details.	Yes
Validate Response Signatures	Check this box to validate the signature of the response from the Identity Provider.	Yes
Validate Assertion Signature	Check this box if it is expected that assertions will be signed. Ignition will validate the signatures of each assertion.	Yes
Signature Verifying Keys	A list of signing key(s) that Ignition uses to validate signatures from the IdP.	Yes
Signature Verifying Certificates	A base64-encoded DER PKIX certificate value.	No

SAML Signature Verifying Key Configuration

Property Name	Description	Required?
Key Algorithm	The algorithm identifier for this signature verifying key. Options are DSA, RSA, or EC.	Yes
Key Value	A base64-encoded DER key value.	Yes

IdP Examples and Troubleshooting

The [OpenID Connect 1.0 Example](#) page will show you how to configure an external IdP that used OpenID Connect 1.0 with your Ignition system. Go to [Troubleshooting Identity Providers](#) for helpful examples to help you diagnose and troubleshoot issues with configuring IdPs. Refer to [SAML Example](#) page for how to configure an Identity Provider that is using the SAML protocol.

In This Section ...

User Attribute Mapping

The User Attribute Mapping page allows you to map information in the Identity Providers (IdP) response document to easily understandable properties. These properties are then made available as [Session Properties](#) in the Perspective Session. To work, this requires that the Gateway already has a valid IdP configuration which returns a response document when attempting to login.

On this page ...

- [Configuring a User Attribute Mapping](#)
- [User Attribute Mapping Property Reference Table](#)
- [Direct Mapping](#)
 - [Attribute Sources](#)
 - [Expression Mapping](#)

Configuring a User Attribute Mapping

1. From the Gateway Webpage **Config** tab, click on **Security > Identity Providers**.
2. Select the **Identity Provider** and click on **User Attribute Mapping** under the **More** button.

The screenshot shows the Ignition Gateway's Configuration interface. In the top navigation bar, the path 'Config > Security > Identity Providers' is selected. Below this, a table lists two identity providers:

Name	Type	Description	Action
default	Ignition	Automatically generated Ignition Identity Provider which uses the User Source Profile named "default".	More Settings
Okta_2018	OpenID Connect 1.0		More Settings

At the bottom of the table, there are two buttons: 'Create new Identity Provider...' and 'Import Identity Provider...'. To the right of these buttons is a vertical menu with several options: 'User Attribute Mapping' (which is highlighted with a red box), 'User Grants', 'Security Level Rules', 'Test Login', 'Export', and 'Delete'. The 'User Attribute Mapping' option is the target of the second step in the configuration process.

3. The Name, Description, and Provider Type are not editable here, but are listed on the page to make clear which IdP the User Attribute Mapping is being configured on.

Main		* Required Field
Provider Name *	Okta_2018 Nickname for the provider.	
Provider Description	Provider Description Description for the provider.	
Provider Type *	OpenID Connect 1.0 The type of the provider.	

4. Under the **User Attributes** section of properties, you'll find settings for each of the mappable user attributes.

Config > Security > Identity Providers > Okta_2018 - User Attribute Mapping

User Attributes		* Required Field
ID *	Type <input type="text" value="direct"/> <input type="text" value="direct"/> <input style="background-color: #0070C0; color: white; border: 1px solid #0070C0;" type="text" value="expression"/> ID Token Claims <input type="text"/> The name of the attribute source.	
	Path <input type="text" value="Path"/> Path to the attribute to map.	
Username *	Type <input type="text" value="expression"/> The type of mapping to apply for usernames. Expression <input type="text" value="Expression"/> The source code for the expression used to map the targeted user attribute.	
First Name	Type <input type="text" value="-None-"/> The type of mapping to apply for first names.	
Last Name	Type <input type="text" value="-None-"/> The type of mapping to apply for last names.	
Email	Type <input type="text" value="-None-"/> The type of mapping to apply for email addresses.	
Roles	Type <input type="text" value="-None-"/> The type of mapping to apply for user roles.	
		<input type="button" value="Cancel"/> <input type="button" value="Save"/>

5. If you make changes to any of the mappings, be sure to click **Save** when finished.

User Attribute Mapping Property Reference Table

Mappings can be configured on several user attributes.

- ID - The Type of mapping to apply for user IDs.
- Username - The Type of mapping to apply for usernames.
- First Name - The type of mapping to apply for first names.
- Last Name - The type of mapping to apply for last names.
- Email - The type of mapping to apply for email addresses.
- Roles - The type of mapping to apply for user roles.

Each attribute mapping has similar properties, which are listed below

Property Name	Description
Type	The Type of mapping to apply for the attribute. Options are direct or expression .
Source	The name of the attribute source. In cases where the identity provider has several Attribute Sources , this property allows you to specify which source the mapping should use. (for direct type only)
Path	Path to the attribute map based on the selected source. Each node in the path is delimited by a slash character ("/"). (for direct type only)
Expression	The source code for the expression used to map the targeted user attribute. See the Security Level Rules . (for expression type only)

Direct Mapping

Direct mappings require that you enter in the path to an attribute in the response document for the given property. Values in the Identity Provider's response document are dereferenced using the configured paths and are copied to the respective user properties in Ignition. The Source dropdown allows a particular mapping to target a specific object in the response document. The Path property then determines what object inside of source should be.

In the image below, the ID of the user, as represented on the Ignition Gateway, will be determined by the **sub** attribute, located in the **ID Token Claims** source in the response document.

The screenshot shows a configuration dialog titled "User Attributes". It contains three main sections: "Type" (set to "direct"), "Source" (set to "ID Token Claims"), and "Path" (set to "sub"). The "Type" section includes a description: "The type of mapping to apply for user IDs." The "Source" section includes a description: "The name of the attribute source." The "Path" section includes a description: "Path to the attribute to map."

Attribute Sources

Each Identity Provider type has a different list of possible source items:

Ignition IdP

- ID Token Claims (default)
- Token Endpoint Response

OIDC

- ID Token Claims (default)
- Token Endpoint Response
- User Info Claims

Note: The user info claims attribute source is present only when the user info URL is configured and Ignition receives a valid user info response from the IdP.

SAML

- Authentication Response

Expression Mapping

The Expression type allows you to use the expression language to derive the attribute from contextual data such as the IdP response document or Tags.

Username *	Type expression
	The type of mapping to apply for usernames.
	Expression <input type="text" value="Expression"/>
	The source code for the expression used to map the targeted user attribute.

Generally this type of mapping is used in conjunction with the built-in `attribute-source` object, which yields the full response. From there, the rest of the expression language can be used to parse the document looking for specific values.

For example, if the response looked like the following:

```
"idTokenClaims": {
  "name": "John Person"
}
```

Then a mapping could be configured with the following expression:

```
{attribute-source:idTokenClaims:name}
```

Furthermore, you could use other functions and syntax in the expression language to further manipulate the results, such as returning just the first name:

```
split({attribute-source:idTokenClaims:name}, ' ')[0,0]
```

First Name	Type expression
	The type of mapping to apply for first names.
	Expression <input type="text" value="split({attribute-source:idTokenClaims:name}, ' ')[0,0]"/>
	The source code for the expression used to map the target attribute.

OpenID Connect 1.0 Example

This section provides an example of how to connect an Identity Provider that is using the OpenID Connect 1.0 protocol. This example uses the Okta IdP service. Your IdP vendor may differ and the specific links will differ.

Prerequisites

Before you begin configuring Ignition there are some preliminary requirements that need to be done outside of Ignition:

- A configured remote IdP (Okta in this example)
- The metadata file specific to your IdP
- The scope data specific to your IdP
- Login credentials to use as a test

On this page ...

- [Prerequisites](#)
 - [Configured IdP](#)
 - [Metadata File](#)
 - [Scope Data](#)
 - [Test Login Credentials](#)
 - [Configure Ignition Gateway](#)

Configured IdP

An IT department is usually the one to setup and configure a remote IdP. You need a configured remote IdP that is compatible with OpenID Connect 1.0. protocol. At minimum there needs to be an account set up with the IdP, users added to the IdP account, and applications added to the IdP.

Metadata File

You will need the metadata file specific to your IdP. This document defines how to communicate with the IdP. It is usually a web page that allows the metadata file to be exported to a JSON file. Often it is a URL that ends with ".well-known/openid-configuration".

You will need the URL link to this page or a JSON export of this page. For example, if your IdP user login URL is something like this:

<https://dev-123456.oktapreview.com/login/login.htm>

Then the metadata import URL may look like something like this:

<https://dev-123456.oktapreview.com/.well-known/openid-configuration>

Here is an example of part of a metadata file for Okta. Notice that the URL link has "/.well-known/openid-configuration" at the end and is very similar to the login URL. It is recommended to use the URL specific to your IdP. Your IT department may choose to export this JSON file of this page and provide it to you. Either option will work.

```

{
  "issuer": "https://dev-123456.oktapreview.com",
  "authorization_endpoint": "https://dev-123456.oktapreview.com/oauth2/v1/authorize",
  "token_endpoint": "https://dev-123456.oktapreview.com/oauth2/v1/token",
  "userinfo_endpoint": "https://dev-123456.oktapreview.com/oauth2/v1/userinfo",
  "registration_endpoint": "https://dev-123456.oktapreview.com/oauth2/v1/clients",
  "jwks_uri": "https://dev-123456.oktapreview.com/oauth2/v1/keys",
  "response_types_supported": [
    "code",
    "id_token",
    "code id_token",
    "code token",
    "id_token token",
    "code id_token token"
  ],
  "response_modes_supported": [...],
  "grant_types_supported": [...],
  "subject_types_supported": [...],
  "id_token_signing_alg_values_supported": [
    "RS256"
  ],
  "scopes_supported": [
    "openid",
    "email",
    "profile",
    "address",
    "phone",
    "offline_access",
    "groups"
  ]
}

```

Note: It is recommended to use the URL specific to your IdP. Your IT dept. may choose to export this JSON file of this page and provide it to you. Either option will work.

Scope Data

When a user is verified by the IdP a lot of the user specific data is not returned in the response file by default (i.e., username, email, firstname, lastname, etc). This user specific data is called the scope, and it can be returned if the **Scope** section of the Ignition Gateway is configured. The list of available scope definitions may be available from your IT department or available from the developer documentation of the IdP you are using.

For this Okta example, the scope data is in the developer notes at <https://developer.okta.com/docs/api/resources/oidc#scope-dependent-claims-not-alwaysReturned>

Test Login Credentials

You need an account specific to the IdP for testing purposes (Okta in this example). To test and verify the IdP account, login to your IdP. For our example, the Okta login page is shown here:

Configure Ignition Gateway

1. On the Gateway Webpage, click on the **Config** tab. You will need to log in if you aren't already.
2. Under the Security section, click on **Identity Providers**. The Identity Providers screen is displayed. This screen will list all IdPs that have been configured. You can filter by name or adjust the number of IdPs displayed per page in the view.



3. Click on the **Create a New Identity Provider...** link.

A screenshot of the Ignition Identity Providers list screen. At the bottom left, there is a blue link labeled '→ Create new Identity Provider...'. This link is highlighted with a red box.

4. Select the **OpenID Connect 1.0** option and click **Next**.

A screenshot of the 'Create New' identity provider configuration screen. It shows three options: 'Ignition', 'OpenID Connect 1.0', and 'Security Assertion Markup Language 2.0'. The 'OpenID Connect 1.0' option is selected and highlighted with a red box. Below the options, there is descriptive text about OpenID Connect 1.0. At the bottom, there are 'Cancel', 'Step 1 of 2', and 'Next' buttons.

5. On the Basic Details screen, provide an **Provider Name**. You can also add an **Provider Description** if desired. The Provider Type field will fill in automatically from the previous screen.

Config > Security > Identity Providers > Create New

Basic Details		* Required Field
Provider Name *	Okta_Example Give the provider a name.	
Provider Description	Test setup for Okta IdP. A description for the provider.	
Provider Type *	OpenID Connect 1.0 The type of the provider.	

6. The next section is **Import Provider Metadata**. In the **Import from URL** section, enter in the URL from earlier that shows the ".well-known/openid-configuration" link specific to your IdP. You can also import a file below if it was provided by your IT department.
7. Click on the **Import** button.

Import Provider Metadata	
Import from URL	<input type="text" value="https://dev.123456.oktapreview.com"/> URL to the OpenID Provider Configuration document. Typically, if the issuer is "https://example.org/foo" then the metadata URL would be "https://example.org/foo/.well-known/openid-configuration" <input style="background-color: #007bff; color: white; border: none; padding: 5px;" type="button" value="Import"/>
Import from File	<input type="button" value="Choose File"/> No file chosen File must be a JSON document with the properties described in section 3 (OpenID Provider Metadata) of the OpenID Connect Discovery 1.0 specification. <input style="background-color: #007bff; color: white; border: none; padding: 5px;" type="button" value="Import"/>

8. Ignition will now generate a URI redirect address for your Ignition server. It is listed just below the "Import Provider Metadata" area of the configuration page.
 In our example it is <http://10.10.115.3:8088/data/federate/callback/oidc>. You need to provide this URI to your IdP (usually this means giving it to your IT department).

Note: The URI should be a web address that is accessible from the IdP server.

💡 Most OpenID Providers will require registering Ignition as a client. After the registration process is complete, the provider will generate a client ID and secret for Ignition, which is required below. This gives Ignition the ability to communicate securely with the provider. Most providers will also require a set of redirect URIs. The redirect URI for this Ignition Gateway is:

http://localhost:8088/data/federate/callback/oidc

9. Once you've giving your IT department the redirect address, they can add your Ignition server as an application to the IdP. Once they have done this, they can provide you with a "Client ID" and "Client Secret". This is needed for the Ignition Gateway to properly communicate with the IdP.

Note: The IdP can use the same redirect address for the Login, Logout, and Initiate Login.

10. The next section is Provider Configuration. Most of the fields below should now be filled in when you imported the IdP Metadata. Fill in the **Client ID** and **Client Secret** fields with the information obtained from your IdP (or IT department). If you don't know them yet, you can put in

bogus values for now and edit them later once the correct values are provided to you.

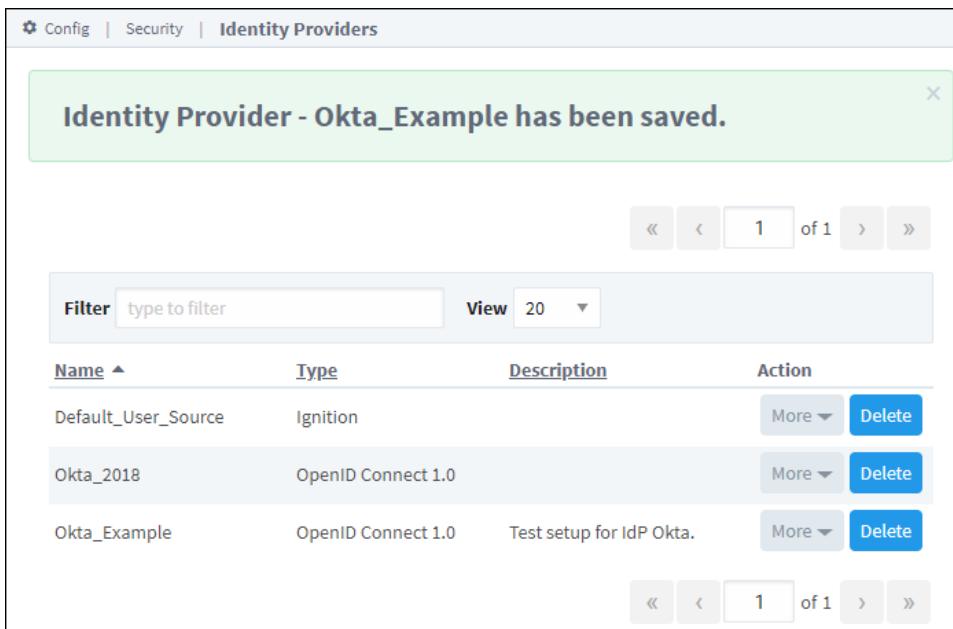
Provider Configuration		* Required Field				
Client ID *	ooaifp0dlnt1234560h7 The client identifier registered within the identity provider.					
Client Secret *	Client Secret The client secret registered within the identity provider. <input type="checkbox"/> Check this box to change the client secret					
Authorization URL *	https://dev-123456.oktapreview.com/ URL of the OP's OAuth 2.0 Authorization Endpoint.					
Token URL *	https://dev-123456.oktapreview.com/ URL of the OP's OAuth 2.0 Token Endpoint.					
Logout URL	https://dev-123456.oktapreview.com/ Optional URL at the OP to which an RP can perform a redirect to request that the End-User be logged out at the OP.					
Json Web Keys URL	https://dev-123456.oktapreview.com/ URL of the OP's JSON Web Key Set document. <input checked="" type="checkbox"/> Use Json Web Keys URI If checked, then identity provider public keys will be automatically downloaded from given Json Web Keys URL. New keys will be automatically fetched when the identity provider generates new keys. If unchecked, then the static set of Json Web Keys (configured below) are used, so when the identity provider rotates keys, they must be manually added to this configuration.					
Issuer *	https://dev-123456.oktapreview.com/ Entity that issues a set of Claims.					
Supported ID Token Signing Algorithm Values	-None- A list of the JWS signing algorithms supported by the OP for the ID Token to encode the Claims in a JWT. <table border="1"> <thead> <tr> <th>Name</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>RS256</td> <td>Remove</td> </tr> </tbody> </table>		Name	Action	RS256	Remove
Name	Action					
RS256	Remove					
Scope	Scope	Add A list of default scopes which will be sent for each auth request to the OP.				
JSON Web Key Config	A list of signing key(s) the RP uses to validate signatures from the OP. <table border="1"> <thead> <tr> <th>Name</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>W4IYgV7xggajMkl5T1234561234567Of-SsKViOM_Jal</td> <td>Edit Remove</td> </tr> </tbody> </table> <p>→ Create new JSON Web Key</p>		Name	Action	W4IYgV7xggajMkl5T1234561234567Of-SsKViOM_Jal	Edit Remove
Name	Action					
W4IYgV7xggajMkl5T1234561234567Of-SsKViOM_Jal	Edit Remove					

11. Providing scope is optional. These fields are specific to your IdP, and you may need to find the developer documentation specific to your IdP.

Enter "email" in the **Scope** field and press the **Add** button. Repeat for each scope you want returned. For our example, the list of available scopes is in the Okta developer documentation: <https://developer.okta.com/docs/api/resources/oidc#scope-dependent-claims-not-alwaysreturned>.

Scope	<input type="text" value="email"/> Add A list of default scopes which will be sent for each auth request to the OP.				
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #333; color: white;">Name</th> <th style="background-color: #333; color: white;">Action</th> </tr> </thead> <tbody> <tr> <td>profile</td> <td style="text-align: right;">Remove</td> </tr> </tbody> </table>	Name	Action	profile	Remove
Name	Action				
profile	Remove				

12. Press the **Save** button at the bottom right of the page. You'll see a confirmation message.

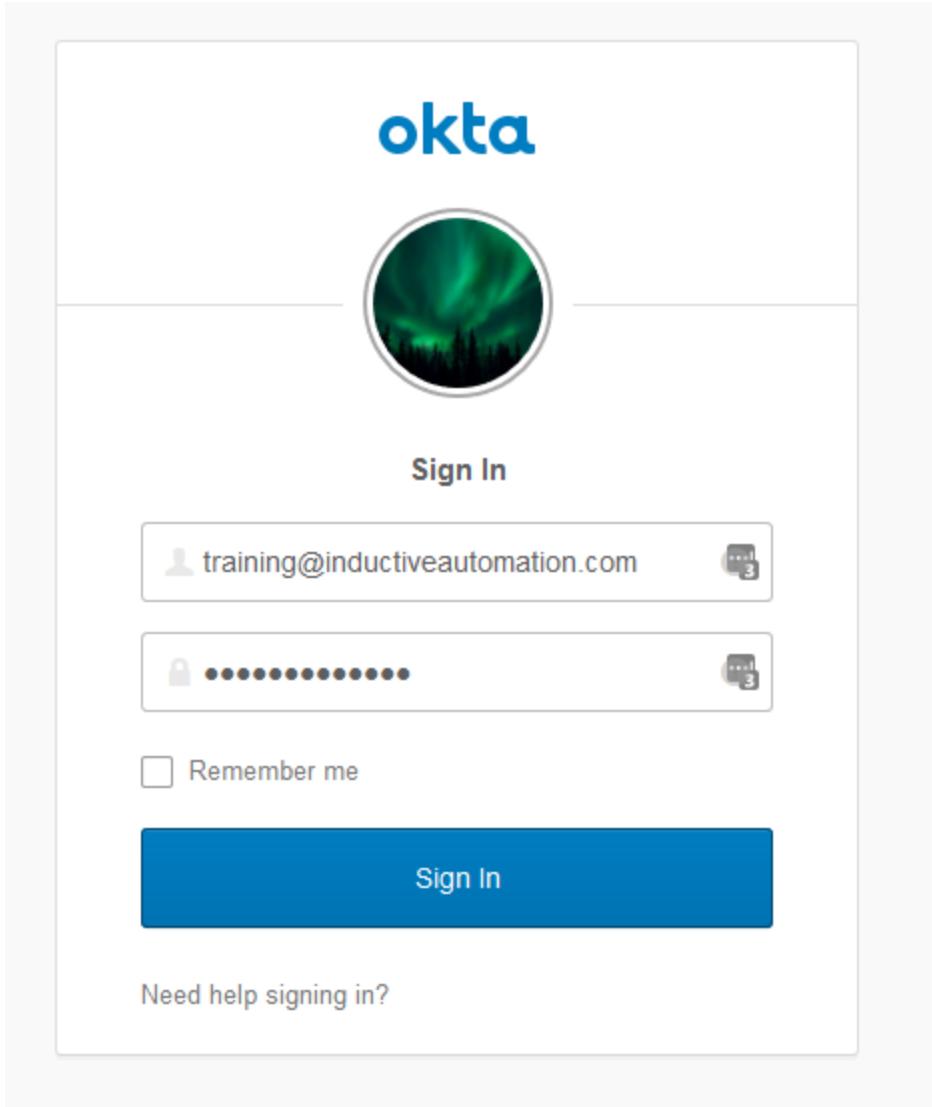


The screenshot shows the Ignition configuration interface under the 'Identity Providers' tab. A green confirmation box at the top center says 'Identity Provider - Okta_Example has been saved.' Below it is a table listing three identity providers:

Name	Type	Description	Action
Default_User_Source	Ignition		More Delete
Okta_2018	OpenID Connect 1.0		More Delete
Okta_Example	OpenID Connect 1.0	Test setup for IdP Okta.	More Delete

13. The next step is to perform a test login. From the Identity Providers screen, select **More** and then **Test Login**.

14. You will be re-directed to the Okta login. Enter in your test login credentials and click the **Sign In** button.



15. If the login is successful, you will be returned to the Identity Provider Test Login screen. The returned results will be displayed in the Results section.

Note: In this example, we did not add the username or email to our scope. Thus they have not been returned.

Test Login	
Adapter Name	Okta_Example Nickname for the adapter.
Adapter Description	Adapter Description Description for the adapter.
Provider Type	OpenID Connect 1.0 The type of the adapter.
Results	{ "sub": "00uifxlfn2yjSXQzC0h7", "ver": 1, "iss": "https://dev-997763.oktapreview.com/oauth2/default", "aud": "0oaifp0dlntvhYxaX0h7", "iat": 1546555091, "exp": 1546558691, "jti": "ID.tTIUVnMSyWGzHyCsBf8vf95XdZN9HCzShEps08b2L5U", "amr": ["pwd"], "idp": "00oiezm8tjcSqDHJM0h7", "nonce": "D11EhKChylzTog2i36FNsqU_PEMCod946X1rfyo52mE", "auth_time": 1546555004, "at_hash": "yOe0W--oTOxdWeYC2s8M8w" }

SAML Example

This section provides an example of how to connect an Identity Provider that is using the SAML protocol. This example uses the Okta IdP service. Your IdP vendor may differ and the specific links will differ.

Prerequisites

Before you begin configuring Ignition there are some preliminary requirements that need to be done outside of Ignition:

- A configured remote IdP (Okta in this example)
- The metadata file specific to your IdP
- The scope data specific to your IdP
- Login credentials to use as a test

On this page ...

- [Prerequisites](#)
 - [Configured IdP](#)
 - [Metadata File](#)
 - [Test Login Credentials](#)
 - [Configure Ignition Gateway](#)

Configured IdP

An IT department is usually the one to setup and configure a remote IdP. You need a configured remote IdP that is compatible with SAML protocol.

At minimum there needs to be an account set up with the IdP, users added to the IdP account, and applications added to the IdP.

Metadata File

You will need the metadata file specific to your IdP. This document defines how to communicate with the IdP. It is usually a web page that allows the metadata file to be exported to an XML file.

You will need the URL link to this page or an XML export of this page. For example, the metadata import URL may look like something like this:

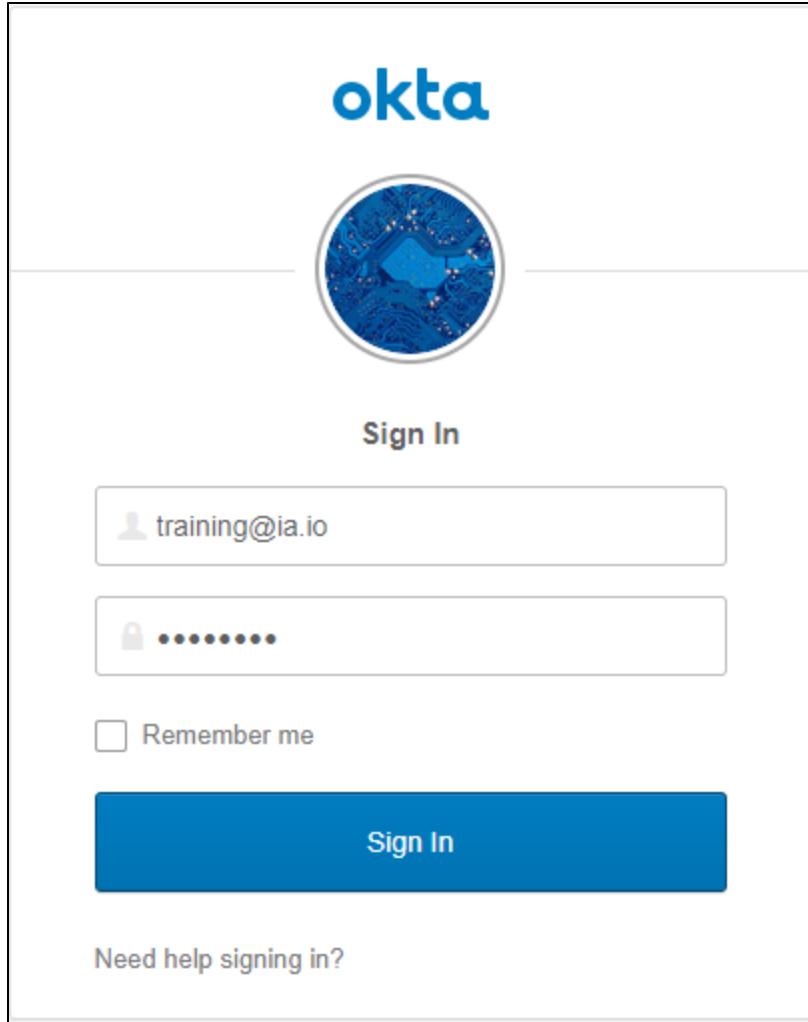
```
https://dev-123456.okta.com/app/esdfdsdf7886sd6723hjkdf/sso/saml/metadata
```

Here is an example of part of a metadata file for Okta. Notice that file is XML format. You can use the file or the URL to automatically import the configuration into Ignition. Otherwise it will need to be manually typed in.

```
<?xml version="1.0" encoding="UTF-8"?>
- <md:EntityDescriptor entityID="http://www.okta.com/exkinzujwtqNw0fzI356" xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata">
  - <md:IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol" WantAuthnRequestsSigned="false">
    - <md:KeyDescriptor use="signing">
      - <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        - <ds:X509Data>
          <ds:X509Certificate>MIIDpDCCAoygAwIBAgIGAWppx7LZMA0GCSqGSIb3DQEBCwUAMIGSMQswCQYDVQQGEwJVUzETMBEG
A1UECAwKQ2FsaWZvcm5pYTEwMBQGA1UEBwwNU2FuIEZyYW5jaXNjbzENMAgA1UECgwET2t0YTEU
MBIGA1UECwvLU1NPUHJvdmlkZXIxEzARBgNVBAMMCmRldi1MzAzMDIxHDAABgkqhkiG9w0BCQEW
DWluZm9Ab2t0YSj20wHhcNMTkwNDI5MTU0NjM2WhcNMjkwNDI5MTU0NzM2WjCBkjELMAkGA1UE
BhMCVVMeEzARBgNVBAgMCkNhbgImb3JuaWExFjAUUBgNVBACMDVNhbIBGcmFuY2lY28xDTALBgNV
BBoMBE9rdGExFDASBgNVBAAsMC1NTT1Byb3ZpZGVyMRMwEQYDVQDDApkZXxtNTMwMzAyMRwwGgYJ
KoZlhvcNAQkBFg1pbmZvQG9rdGEyANBgkqhkiG9w0BAQEFAAOCAQ8AMIBCgKCAQE
26clu1q2eNnUp4VBhnsJ5mucEbpoSReDzaG8/CLNj73mp5dCicLZ/zrw/6SO+VbIV2drShOJDQzTx
sBfPOCm7cn2QV8CARW8Ci/oPPnqrXp/8yetElp4tea1g7WQbkZxO0uStjRlOrW6ISKgKrWEo1LII
6iU7kdbu7GIUUhY6KpSuMpT3iQq0MiQV7v+VFVx03VpGiAVFGIPY4hbDc0PwURN3MBw4cnhCK
kTsTAqwsSwH0N+9wsjwYWhEEMDYq/Eo0pbA2sGkChkz/vYJSUXjWMKnD432T3tYNYxEm95HzAIJn3
ixdn8IPTbiERH60LyYWQtn55TAf5oCCq4UxGowIDAQABMA0GCSqGSIb3DQEBCwUA4IBAQ4PWYQ
QBdfPnkWx2Tqacsm3BnliPvQaslzejk/9+jL3PKvR/GQIQMFaFis2Fc+ucB426d0u3EbWSB8e9v4
deSVWytwpMEHQijKHV8zjaX2AcWWRgiv/1VR4Klp2cahX3h2tU+n87A3a3su201VuJ+inSIxquTu
```

Test Login Credentials

You need an account specific to the IdP for testing purposes (Okta in this example). To test and verify the IdP account, login to your IdP. For our example, the Okta login page is shown here:



You should now have a IdP credentials to test with, a metadata URL or metadata XML file. The next step is to configure Ignition to communicate with your IdP.

Configure Ignition Gateway

1. On the Gateway Webpage, click on the **Config** tab. You will need to log in if you aren't already.
2. Under the Security section, click on **Identity Providers**. The Identity Providers screen is displayed. This screen will list all IdPs that have been configured. You can filter by name or adjust the number of IdPs displayed per page in the view.



3. Click on the **Create a New Identity Provider...** link.

The screenshot shows the 'Identity Providers' page under 'Config > Security'. There is one provider listed:

Name	Type	Description	Action
default	Ignition	Automatically generated Ignition Identity Provider which uses the User Source Profile named "default".	More Settings
Okta_2018	OpenID Connect 1.0		More Settings

Below the table are two buttons: 'Create new Identity Provider...' (highlighted with a red box) and 'Import Identity Provider...'.

4. Select the **Security Assertion Markup Language 2.0** option and click **Next**.

The screenshot shows the 'Create New' configuration screen for an identity provider. It has three options:

- Ignition**: The instance of Ignition hosting this configuration can act as an identity provider. This is useful in cases where no external identity provider is available.
- OpenID Connect 1.0**: OpenID Connect 1.0 (OIDC) is an authentication layer on top of OAuth 2.0, an authorization framework.
- Security Assertion Markup Language 2.0**: Security Assertion Markup Language 2.0 (SAML) is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider.

At the bottom are 'Cancel', 'Step 1 of 2', and 'Next' buttons.

5. On the Basic Details screen, provide an **Provider Name**. You can also add an **Provider Description** if desired. The Provider Type field will fill in automatically from the previous screen.

Config > Security > Identity Providers > Create New

Basic Details		* Required Field
Provider Name *	Okta_SAML_Example Give the provider a name.	
Provider Description	Test setup for SAML IdP A description for the provider.	
Provider Type *	Security Assertion Markup Language 2 The type of the provider.	

6. The next section is **Import Provider Metadata**. In the **Import from URL** section, enter in the URL from earlier specific to your IdP. You can also import a file below if it was provided by your IT department.
7. Click on the **Import** button.

Import Provider Metadata

Import from URL	<input type="text" value="https://dev-123456.okta.com/app/esd"/> URL to the SAML Identity Provider Metadata document. Import
Import from File	<input type="button" value="Choose File"/> No file chosen File must be an XML document which conforms to the SAML 2.0 metadata schema described in saml-metadata-2.0-os. Import

8. Ignition will now generate a URI redirect address for your Ignition server. It is listed just below the “Import Provider Metadata” area of the configuration page.
In our example it is <http://10.10.110.86:8088/data/federate/callback/saml>. You need to provide this URI to your IdP (usually this means giving it to your IT department).

Note: The URI should be a web address that is accessible from the IdP server.

 The SP Entity ID for this Ignition Gateway is: **http://localhost:8088**
 The Assertion Consumer Service (ACS) URL for this Ignition Gateway is: **http://localhost:8088/data/federate/callback/saml**
 The SP Metadata for this Ignition Gateway may be accessed at the following URL: **http://localhost:8088/data/saml/metadata/sp**

9. Once you have given your IT department the redirect address, they can add your Ignition server as an application to the IdP.

Note: The IdP can use the same redirect address for the Login, Logout, and Initiate Login.

10. The next section is Provider Configuration. Most of the fields below should now be filled in when you imported the IdP Metadata.

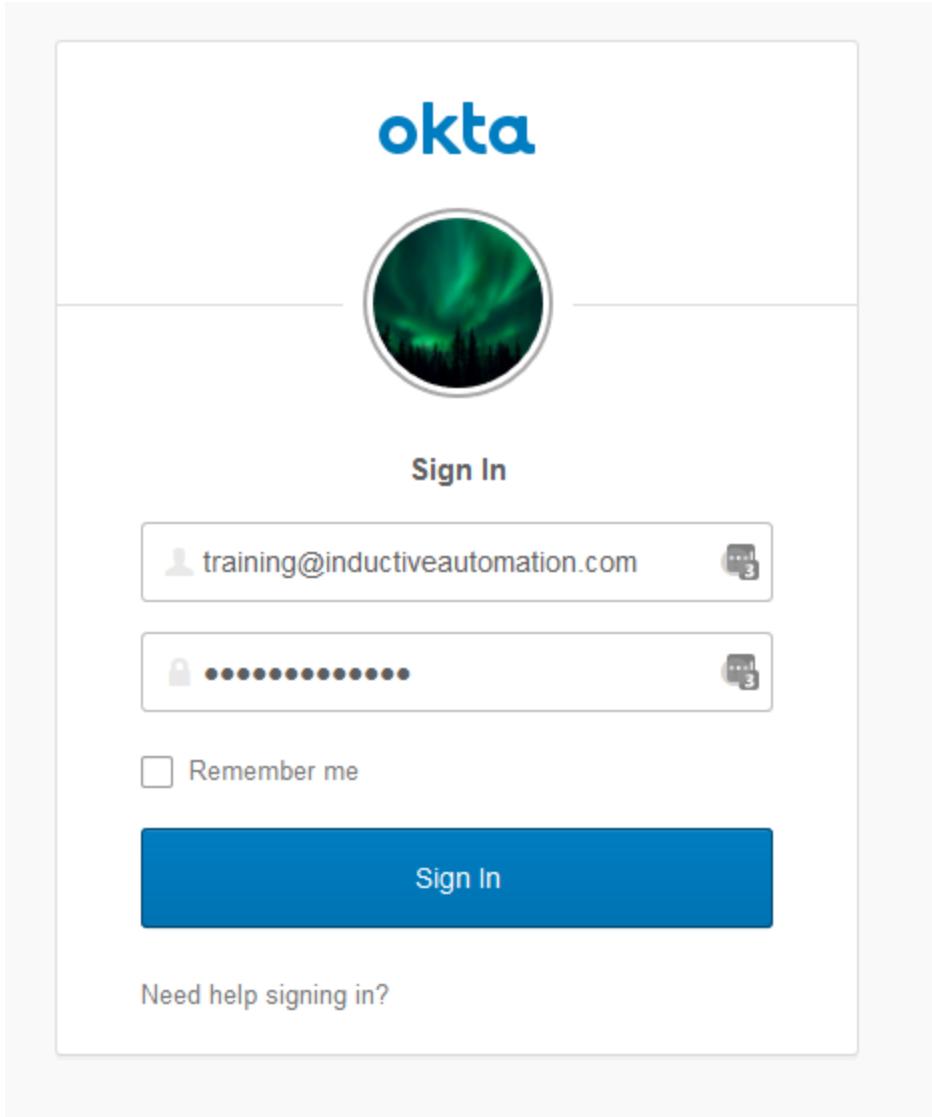
Provider Configuration		* Required Field
Entity ID *	<input type="text" value="http://www.okta.com/exkinzujwtqNwl"/> The Identity Provider's Entity ID.	
Assertion Consumer Service (ACS) Binding *	<input type="text" value="HTTP_POST"/> <p>The expected binding used by the Identity Provider when interacting with Ignition's Assertion Consumer Service.</p>	
Name ID Format *	<input type="text" value="UNSPECIFIED"/> <p>The expected name ID format for subjects of assertions resulting from Authn Requests.</p>	
Single Sign-On (SSO) Service URL *	<input type="text" value="https://dev-530302.okta.com/app/indi"/> <p>The Identity Provider's Single Sign-On (SSO) Service URL.</p>	
Single Sign-On (SSO) Service Binding *	<input type="text" value="HTTP_POST"/> <p>The binding Ignition will use for sending Authn Requests to the Identity Provider's Single Sign-On (SSO) Service.</p>	
Force Authn *	<input type="checkbox"/> Check this box to force complying Identity Providers to authenticate the user each time instead of relying on a previous security context. See section 3.4.1 of saml-core-2.0-os for more details.	
Validate Response Signatures *	<input checked="" type="checkbox"/> Check this box to validate the signature of the response from the Identity Provider.	
Validate Assertion Signatures *	<input type="checkbox"/> Check this box if it is expected that assertions will be signed. Ignition will validate the signatures of each assertion.	
Signature	A list of signing key(s) that Ignition uses to validate signatures from the IdP.	

11. Press the Save button at the bottom right of the page. You'll see a confirmation message.

The screenshot shows the Ignition configuration interface under the 'Config > Security > Identity Providers' path. A green success message box is displayed, stating 'Identity Provider - Okta_SAML_Example has been saved.' Below the message is a table listing three identity providers:

Name	Type	Description	Action
default	Ignition	Automatically generated Ignition Identity Provider which uses the User Source Profile named "default".	<button>More</button> <button>Delete</button>
Okta_SAML_Example	Security Assertion Markup Language 2.0	Test setup for SAML IdP	<button>More</button> <button>Delete</button>
opcua-module	Ignition	Automatically generated Ignition Identity Provider which uses the User Source Profile named "opcua-module".	<button>More</button> <button>Delete</button>

12. The next step is to perform a test login. From the Identity Providers screen, select **More** and then **Test Login**.
 13. You will be re-directed to the Okta login. Enter in your test login credentials and click the **Sign In** button.



14. If the login is successful, you will be returned to the Identity Provider Test Login screen. The returned results will be displayed in the Results section.

Test Results - Security Assertion Markup Language 2.0

[IdP Response Data](#) [Mapped User Attributes](#) [Security Level Grants](#)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<saml2p:Response xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" Destination="http://10.10.110.86:8088/d
<saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Format="urn:oasis:names:tc:SAML:2.0:na
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
        <ds:Reference URI="#id33568854137229631961225447">
            <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
            <ds:DigestValue>voI6f9hmc3ObTzHTm+onwR6QdmPaPF4zYFLSs+7Qous=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>ijLjE5ZiGe1VVxf3BWPF2naBrYEmaHoaHooESPiEcsVlRvMLnuoijmJ9ZUMCn+iqcAsCW+UbYdkamtPro
    <ds:KeyInfo>
```

User Grants

A User Grant is a way to directly assign a user to a [Security Level](#), even if they do not meet the requirements of the [Security Level Rules](#). User Grants essentially act as an override to the original rules of the Security Level.

User Grants are accessed from the Gateway Webpage **Config** section in **Security > Identity Providers**. Users can be added and edited using the buttons in the Users table so that Security Levels can then be granted to them.

Note: When adding, editing, and deleting users in User Grants, you are only modifying the User Grant (whether the user is granted permission that overrides the Security Rules). The user is not changed in the Identity Provider.

Users are identified by either their username or their ID from the provider response document. Once you have identified a user, you can assign them any number of grants to Security Levels. Selecting a level will automatically select all security levels above it. The User Grants can only be applied to a user after they authenticate with the Identity Provider, though the grants do not have to be for levels within the Authenticated branch.

Note: The system can't validate any user created here against actual users in the Identity Provider (IdP). Instead, the username or ID needs to be entered exactly, and when a user logs in, the system will check to see if they match any of the configured usernames/IDs to give User Grants to.

On this page ...

- [Configuring a User Grant](#)



INDUCTIVE
UNIVERSITY

User Grants

[Watch the Video](#)

Configuring a User Grant

There are two parts to configuring a User Grant: Adding a user then applying User Grants.

1. From the Gateway Webpage **Config** tab, click on **Security > Identity Providers**. The screen will refresh and you will see a list of all your IdPs.
2. Choose the IdP and click the **More** button to see the actions in the dropdown list.
3. Select **User Grants**.

The screenshot shows the 'Identity Providers' configuration screen. At the top, there are tabs for 'Config', 'Security', and 'Identity Providers'. Below the tabs is a search bar labeled 'Filter' with a placeholder 'type to filter' and a 'View' dropdown set to '20'. A table lists four identity providers: 'Default_User_Source' (Ignition), 'Okta_2018' (OpenID Connect 1.0), 'Okta_Example' (OpenID Connect 1.0), and 'Okta_TroubleShooting' (OpenID Connect 1.0). Each row has an 'Action' column with a 'More' dropdown and a 'Delete' button. The 'More' dropdown for 'Okta_Example' shows 'User Attribute Mapping' and 'User Grants'. The 'More' dropdown for 'Okta_TroubleShooting' shows 'User Grants', 'Security Level Rules', 'Test Login', and 'Export'. At the bottom of the table, there are links for 'Create new Identity Provider...' and 'Import Identity Provider...'. A red box highlights the 'User Grants' option in the 'More' dropdown for 'Okta_TroubleShooting'.

4. To add a new user, click the **Add** icon.
5. Choose how you will identify the user; either with a username or an ID. Click **Confirm** to save the changes.

Add New User

ID
jsmith

Type
 ID Username

Confirm

Cancel

6. With the user created and highlighted in the Users table, select Security Levels to grant them when they Authenticate with this Identity Provider.

Users	
ID / Username	Type
jsmith	ID

- Public
- Authenticated
 - Custom Roles
 - PlantA
 - Floor1
 - Intern
 - Manager
 - Operator
 - NewLevel
 - Weekend Access
 - Roles
 - Administrator
 - Operator
 - Weekend Security

7. Click **Save**.

Now you can test this user through the [Test Login and Logout](#) screen to verify the new roles have been assigned.

Test Login and Logout

On the Identity Providers screen you can test a username and password combination against an Identity Provider (IdP).

When you select the Test Login option for your IdP, it will confirm the IdP name and Type that you are testing against. It gives you a way to test your attribute mapping configuration and your security level rules / direct user grants configuration

Clicking the Test Login button will redirect you to the IdP where you can login. Upon successful authentication with the IdP, the page navigates back to Ignition, and Ignition displays the response document as the results. These results can vary between IdPs, so it can be useful to test out a login to see what your IdP returns in its response document.

You can use Test Logout option to log out of the ID you were testing.

Ignition's IdP returns an 'amr' attribute that indicates how the user was authenticated.

- If the user was authenticated with a username and password challenge, the amr returns ["uname", "pwd"]
- If the user was authenticated with a badge challenge, the amr returns: ["badge"]
- If the user was authenticated with a badge and password challenge, the amr returns: [:badge", "pwd"]

With this feature, you can enable different security levels based on how the user authenticated. In a security level rule, you could enter:

```
containsAll ({idp-attributes:amr}, 'uname', 'pwd')
```

On this page ...

- [Test a Login](#)
- [Test a Logout](#)

Test a Login

1. From the Gateway Webpage **Config** tab, go to **Security > Identity Providers**. The window will refresh and your list of Identity Providers will be displayed.
2. Choose the Identity Provider and click the **More** button to see the actions in the dropdown list, and select **Test Login**.

The screenshot shows the Ignition Gateway Webpage with the 'Identity Providers' tab selected. The list of providers includes:

Name	Type	Description	Action
Default_User_Source	Ignition		More ▾ Delete
Okta_2018	OpenID Connect 1.0		More ▾ Test Login Delete
Okta_Example	OpenID Connect 1.0	Test s	More ▾ User Attribute Mapping Delete
Okta_TroubleShooting	OpenID Connect 1.0		More ▾ User Grants Delete

At the bottom of the list, there are links to 'Create new Identity Provider...' and 'Import Identity Provider...'. A red box highlights the 'Test Login' option in the dropdown menu for the 'Okta_2018' provider.

3. Log in at your IdP's login screen.

4. If the login is successful, you will be returned to the Identity Provider Test Login screen. The returned results will be displayed under the IdP Response Data tab.

The screenshot shows the 'Test Results - Ignition' page with the 'IdP Response Data' tab selected. The content area displays a JSON object:

```
{  
  "tokenEndpointResponse": {  
    "access_token": "pE7Z8qg-06EJIrLyi59ceheFHljEZFnM4FEx5dBoieY",  
    "id_token": "eyJraWQiOjIjMSIsImFsZyI6IlJTMyU2In0.eyJpc3MiOiJkZWZhWx0Iiw1YXVkJoi  
    "token_type": "Bearer",  
    "expires_in": 3600,  
    "scope": "openid"  
  },  
  "idTokenClaims": {  
    "iss": "default",  
    "aud": "ignition",  
    "exp": 1602202511,  
    "jti": "pB7xNnbS93vARr_ijgygTQ",  
    "iat": 1602201911,  
    "nbf": 1602201701  
  }  
}
```

5. Click on the Mapped User Attributes tab to view the user attributes for the currently logged in user.

The screenshot shows the 'Test Results - Ignition' page with the 'Mapped User Attributes' tab selected. A red box highlights this tab. Below it is a table showing user attributes:

Attribute Name	Attribute Value
ID	1
Username	admin
First Name	
Last Name	
Email	admin@mycompany.com
Roles	Administrator, Operator

At the bottom are two buttons: 'Test Login' and 'Test Logout'.

6. Click on the Security Level Grants tab to view the Security Levels for the roles of the currently logged in user.

Config > Security > Identity Providers > default - Test Login

Test Results - Ignition

IdP Response Data	Mapped User Attributes	Security Level Grants
-------------------	------------------------	-----------------------

▼ Public
 SecurityZones

▼ Authenticated
 ▼ Roles
 Administrator
 Operator

Test Login **Test Logout**

This screenshot shows the 'Test Results - Ignition' page for the 'default' identity provider. The 'Security Level Grants' tab is selected, indicated by a red box. The interface shows a tree view of security zones and roles. Under 'Public', there is a 'SecurityZones' node. Under 'Authenticated', there is a 'Roles' node with children 'Administrator' and 'Operator'. At the bottom, there are two buttons: 'Test Login' and 'Test Logout'.

Test a Logout

After testing a User ID, you do not want to stay logged in as the user. You can use the Test Logout function to log out. For the Ignition IdP, this function also logs you out of the IdP. For an OpenID Connect IdP, this function will also log you out of the IdP if you have a Logout URL.

1. To log out of the ID you were testing, click the **Test Logout** button on the Test Login page.

Config > Security > Identity Providers > Default_User_Source - Test Login

Test Results - Ignition

IdP Response Data	Mapped User Attributes	Security Level Grants
-------------------	------------------------	-----------------------

▼ Public
 ▼ Authenticated
 ▼ Roles
 Administrator
 Operator
 SecurityZones

Test Login **Test Logout**

This screenshot shows the 'Test Results - Ignition' page for the 'Default_User_Source' identity provider. The 'Test Logout' button is highlighted with a red box. The page structure is identical to the Ignition version above it, with tabs for IdP Response Data, Mapped User Attributes, and Security Level Grants, and a tree view of security zones and roles.

2. You will get a confirmation message of a successful logout.

The screenshot shows the Ignition configuration interface under the 'Security' section, specifically the 'Identity Providers' page. At the top, there is a navigation bar with 'Config > Security > Identity Providers'. Below the navigation is a green success message box with the text 'Successfully logged out of Identity Provider default.' A red rectangle highlights this message box. Below the message is a pagination control with buttons for '«', '<', '1' (highlighted), 'of 1', '>', and '»'. Underneath the controls is a search/filter bar with 'Filter' and 'View 20' dropdowns. The main content area is a table with the following columns: 'Name' (sorted ascending), 'Type', 'Description', and 'Action'. There are two rows in the table:

Name	Type	Description	Action
default	Ignition	Automatically generated Ignition Identity Provider which uses the User Source Profile named "default".	More Settings
Okta_2018	OpenID Connect 1.0		More Settings

At the bottom left of the table area, there are two blue links: 'Create new Identity Provider...' and 'Import Identity Provider...'. Below the table is another set of pagination controls with '«', '<', '1' (highlighted), 'of 1', '>', and '»'.

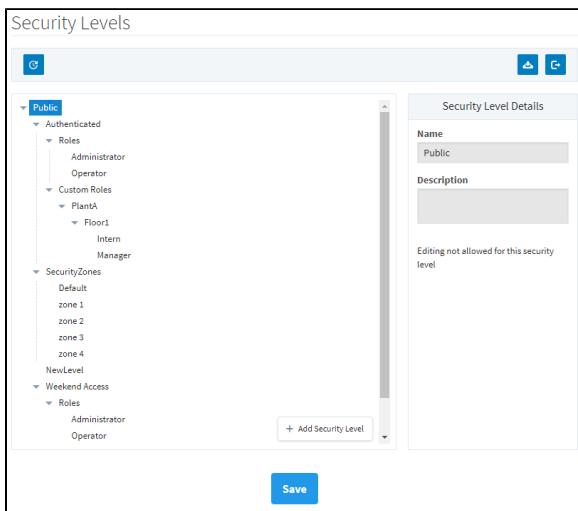
Security Levels

With Security Levels, you define a hierarchy for access inside a Perspective Session or Vision Client. This authorization system provides a way for you to map roles from an Identity Provider (IdP) to Ignition roles. Any IdP can be used to provide roles, and security levels are independent of the type of IdP being used. Any role from the IdP is automatically granted to the user as a role, but only roles in your Security Levels are available to the security screens in the Designer. You can also use the [User Grants](#) option to grant additional access for each user.

Security Levels are defined at the Gateway and they are arranged in a tree structure. Each child (nested) level of the tree inherits the security of its parent levels. There are four reserved Security Levels in the platform:

- Public
- Authenticated
- Authenticated/Roles
- Security Zones

To access Security Levels, go to the Gateway Webpage under the **Config** tab, and choose **Security > Security Levels**.



On this page ...

- Reserved Security Levels
 - Public
 - Authenticated
 - Authenticated/Roles
 - SecurityZones
- Custom Security Levels
- Add a New Security Level
- Edit a Security Level
- Delete a Security Level
- Import a Security Levels Configuration
- Export a Security Levels Configuration



Reserved Security Levels

The reserved security levels are mostly created for you, and have special rules that determine when a user is granted that level. They can't be renamed or deleted.

Public

All users are always granted the Public security level, even if they are not authenticated (logged in). Public security level indicates open access and the least amount of security. A session that only has the Public security level is not authenticated. This is similar to being a guest or anonymous. Unless another security level is required, guest access will be allowed. The Public security level is the ancestor of all other security levels in the hierarchy.

Authenticated

The Authenticated Security Level is a child of the Public Security Level. If a session has authenticated against the configured IdP successfully, the Authenticated Security Level is granted. Users are required to be logged in in order to have access to this level.

Authenticated/Roles

The Roles level is a special level which itself has no special rules, but it acts as a parent placeholder for potential roles returned from the IdP. This particular level is not configurable; however there can be levels added underneath the Roles level as children. These levels should correspond to the names of roles that would be expected from the IdP. If the IdP provides role information, these roles are automatically mapped to the child security levels underneath Authenticated/Roles. The names of the roles must match exactly for them to be correctly mapped to. For example, if you authenticate against the Ignition IdP configured to delegate to the Internal user source, and your user was granted the roles "A", and "B", you would have (at a minimum) the following security levels granted to you:

- Public
 - Authenticated
 - Roles
 - A
 - B

Note: You can only add one level of children to the Roles Security Level. Custom Roles can be nested as deeply as you want.

SecurityZones

The SecurityZones level is another special placeholder level that itself has no rules but is a parent for all of the Security Zones on the Gateway. Security Zones are automatically pulled in from the Gateway. A **Security Zone** is a list of Gateways, Computers, or IP addresses that are defined and grouped together. This group is a zone on the [Gateway Network](#), which can have additional policies and restrictions placed on it. Security Zones provide a way to bridge the IdP method of permissions with location-based permission modeling. You cannot add, edit, or remove the SecurityZones node or any node in the SecurityZones sub-tree.

Custom Security Levels

Custom Security Levels can be added to almost anywhere within the tree. When these levels are granted to a user is determined by the [Security Level Rules](#), which can pull information from the IdP, Security Zones, and even Tags. The placement of custom Security Levels can affect when they may be potentially granted to a user. Any custom levels set under the Public level, but not within Authenticated, do not need to have a user authenticate against the IdP to be granted to a user. However, custom levels within Authenticated do need to have the user authenticate to be granted to the user, even if the rule for that level does not use any of the IdP attributes.

Add a New Security Level

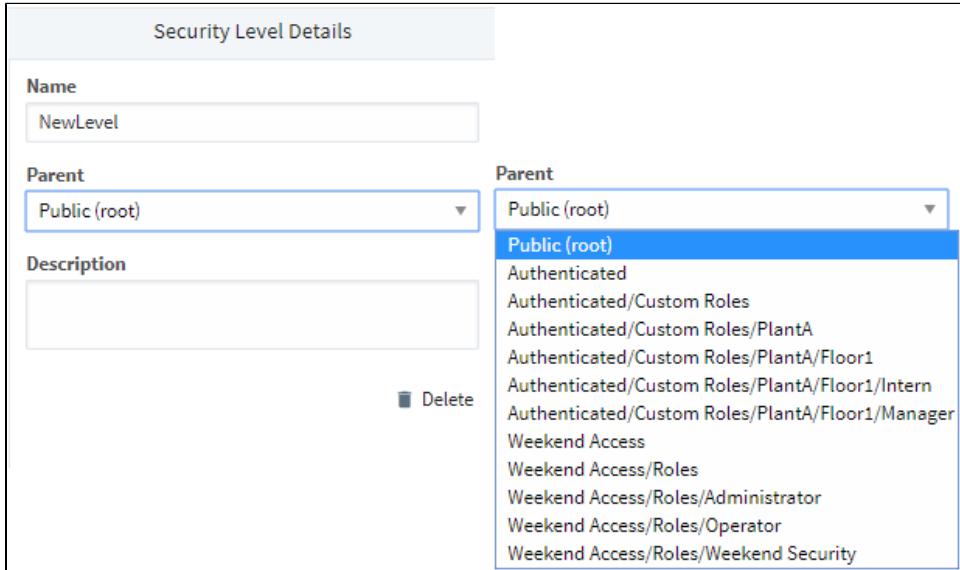
1. From the Gateway Webpage **Config** tab, click on **Security > Security Levels**.
2. In the Security Level tree, select the level that will be a parent for the new level.
3. Click the **Add Security Level** button.

 Add Security Level

4. In the **Security Level Details** screen area, enter the **Name** for the level.

Note: Security Level names within the same parent must be unique.

5. The path for the parent is filled in automatically. Use the **Parent** dropdown list if you want to change the parent for this new level.



The screenshot shows the 'Security Level Details' dialog box. The 'Name' field contains 'NewLevel'. The 'Parent' dropdown is set to 'Public (root)'. To the right, a larger dropdown menu also lists 'Public (root)' at the top, followed by a long list of security levels including 'Authenticated', 'Authenticated/Custom Roles', 'Authenticated/Custom Roles/PlantA', etc.

6. Add a **Description** for the new level (optional).
7. If you make changes to Security Levels, but decide not to save them, you can use the Reset button  to return the tree view to its currently saved configuration.
8. To save the changes, press **Save**.

Edit a Security Level

1. From the Gateway Webpage **Config** section, click on choose **Security > Security Levels**.
2. In the Security Level Tree, select the level that you want to edit.

The screenshot shows the 'Security Levels' screen. On the left is a tree view of security levels under 'Public'. A node 'Manager - West' is selected and highlighted in blue. The tree includes categories like 'Authenticated', 'Custom Roles', 'SecurityZones', and 'Custom'. On the right is a 'Security Level Details' panel. It contains fields for 'Name' (set to 'Manager - West'), 'Parent' (set to 'Authenticated/Custom Roles/PlantA/Floor1'), and 'Description'. Below the panel are buttons for 'Delete' and 'Save'. At the bottom of the screen is a large 'Save' button.

3. Make the desired changes in the **Security Levels Details** screen area.

4. If you make changes to Security Levels, but decide not to save them, you can use the Reset button  to return the tree view to its currently saved configuration.
5. To save the changes, click the **Save** button.

Delete a Security Level

Caution: When you delete a Security Level, all children under that level will also be deleted.

1. From the Gateway Webpage **Config** tab, click on **Security > Security Levels**.
2. In the Security Level tree, select the level that you want to delete.
3. Click the **Delete** button on the **Security Level Details** screen.

Security Level Details

Name
Child Item

Parent
~Public/Authenticated...

Description
Area to provide a description about a specific Security Level.

 Delete

4. In the confirmation box, click **Delete** to confirm the delete.

Import a Security Levels Configuration

1. From the Gateway webpage **Config** tab, click on **Security > Security Levels**.
2. Click the Import icon .
3. Choose **Import** on the confirmation screen.

Import Security Levels Config

Are you sure you want to import a new Security Levels configuration?

Import

Cancel

4. Choose a security levels configuration file to import.
5. Click **Open**.

Export a Security Levels Configuration

1. From the Gateway Webpage **Config** tab, choose **Security > Security Levels**.
2. Click the Export icon .

3. The security levels configuration will be saved as a .json file with a unique number, for example:

 **security-levels-config-1544547744394.json**

Security Level Rules

When a user accesses a project, they can fall into one or more of many [Security Levels](#) that you can set up. The Security Level Rules determine if the user has that Security Level or not. The rules come in the form of expressions that can access any of the Expression Language's functions, Tag values, Security Zone information, or Identity Provider attributes. Security Level Rules are accessible from the Gateway Webpage [Config](#) tab in **Security > Identity Providers**.

Predefined Rulesets

One thing you may notice right away is that on the Security Level Rules page, many of the built in Security Levels are missing. Some Security Levels don't allow you to create an expression that defines their rules. These particular Security Levels already have a set of rules that govern how a user gets them. The Security Levels corresponding to the different Security Zones are automatically given to users depending on which zones they fall into. Similarly, the Security Levels that correspond to the Roles a user gets while authenticated are automatically given to users depending on what roles we receive from the Identity Provider. These Security Levels are removed from the tree here because rules can't be defined on them. The "Public" and "Authenticated" Security Levels also can't have Security Level Rules defined on them, but are present in the tree because they can potentially be a parent to custom nodes which can have Security Level Rules. The Public level is granted to every user when they open the project, and the Authenticated level is granted when the user authenticates against an Identity Provider, regardless of what roles they may have.

On this page ...

- [Predefined Rulesets](#)
- [Defining Security Level Rules](#)
 - [Special Object Reference](#)
 - [Special Function Reference](#)
 - [Evaluating Tag Values](#)
 - [Special Considerations for Rules](#)
- [Configuring Security Level Rules](#)



INDUCTIVE
UNIVERSITY

Security Level Rules

[Watch the Video](#)

Security Levels

The screenshot shows the 'Security Levels' configuration page. On the left, a tree view displays the security level structure for the 'Public' level:

- Authenticated**:
 - Roles**: Administrator, Operator
 - Custom Roles**:
 - PlantA**:
 - Floor1**: Intern, Manager- SecurityZones**: Default, zone 1, zone 2, zone 3, zone 4, NewLevel
- Weekend Access**:
 - Roles**: Administrator, Operator

A button labeled '+ Add Security Level' is located at the bottom right of the tree view area.

On the right, the 'Security Level Details' panel is shown, containing the following fields:

- Name**: Public
- Description**: (empty)

A note below states: "Editing not allowed for this security level".

A large blue 'Save' button is positioned at the bottom center of the page.

Defining Security Level Rules

For the Security Levels that can have rules defined, the rules are defined in the form of an expression which should return either True or False, the results of which determine whether a user falls into that level or not. The rules can take advantage of everything the [expression language](#) has to offer, including the built in expression functions and any Tag values. The expressions here also have the unique ability to access attributes from the Identity Providers response document for the authenticated user, as well as what Security Zone the user falls into.

Special Object Reference

These special objects can be used to reference information gathered from the IdP response document or the Security Zone that the user falls under.

Object Name	Description
{security-zones}	This object gives the collection of security zones that the user currently has. The collection can be handled using one of the unique functions: containsAll or containsAny.

{attribute-source:X:Y} This object can be used to access values in deeper structures by using colons to delimit each object. The [test login page](#) provides a practical way to examine the response document and determine a path to the desired attributes.

Assuming the following JSON response from an identity provider:

```
{
  "idTokenClaims":
  {
    "foo" : "bar"
  },
  "userInfo":
  {
    "email" : "person@place.com"
  }
}
```

{attribute-source:userInfo:email} would retrieve the value of the user's email.

Attribute	Path	Example
ID	sub	{attribute-source:idTokenClaims:sub}
Username	preferred_username	{attribute-source:idTokenClaims:preferred_username}
First Name	given_name	{attribute-source:idTokenClaims:given_name}
Last Name	family_name	{attribute-source:idTokenClaims:family_name}
Email	email	{attribute-source:idTokenClaims:email}
Roles	roles	{attribute-source:idTokenClaims:roles}

Note: The {attribute-source:X:Y} object is only available to Security Levels that fall within the Authenticated Security Level. See below under Special Considerations for Rules.

Special Function Reference

When writing an expression to determine Security Level Rules, there are functions available that are not a part of the normal set available to Expression Bindings. These additional functions are:

Function Name	Description	Example
containsAll(collection, element 0, ..., element N)	Checks to see if all of the listed elements are present in the collection object. The function requires at least two arguments, a collection and an element.	<pre>containsAll({security- zones}, 'PlantA', 'Floor1',</pre>

		'Press Room')
containsAny(collection, element 0, ..., element N)	Checks to see if any of the listed elements are present in the collection object. The function requires at least two arguments, a collection and an element.	containsAny({attribute- source:idTokenClaims: roles}, 'Manager', 'Operator'))

Evaluating Tag Values

Tag values can be accessed in the Security Level Rules expression area by encasing the Tag Path (including the Tag Provider) in braces ("{}")

```
{[tagProvider]path/to/tag}
```

Note: Security Levels are determined on initial login for each session, so if a Security Level is using an expression that references a Tag value, changing the value while the session is running won't change the Security Levels applied to the users already logged in.

Special Considerations for Rules

When defining rules for a Security Level, it is important to notice where in the Security Level tree you are. If you want to access information out of the Identity Provider such as the username, you will need to ensure that the Security Level is located in the Authenticated branch. User information is only captured once a user logs in, so that information will only fall under Security Levels that come from the user being Authenticated. If a Security Level lies outside of the Authenticated branch, then the level will only have access to information such as Tag values and Security Zones.

Configuring Security Level Rules

- From the Gateway Webpage under the **Config** tab, go to **Security > Identity Providers**.
- A list of the Identity Providers will be displayed. Click the **More** button for the Identity Provider you want to edit, and select **Security Level Rules**.

Config | Security | Identity Providers

Name	Type	Description	Action
Default_User_Source	Ignition		More Delete
Okta_2018	OpenID Connect 1.0		Settings More Delete
Okta_Example	OpenID Connect 1.0	Test s	User Attribute Mapping More Delete
Okta_TroubleShooting	OpenID Connect 1.0		User Grants More Delete
			Security Level Rules More Delete

[Create new Identity Provider...](#)
[Import Identity Provider...](#)

3. Select the **Security Level Name** and, if a rule is defined, it will appear in the Rule field. If not, you can create one. We copied the expression "containsAny([security-zones], 'PlantA', 'Floor1', 'Press Room')".

The screenshot shows the Ignition configuration interface. On the left, there is a tree view of security roles:

- Public
 - Authenticated
 - Custom Roles
 - PlantA
 - Floor1
 - Intern (highlighted in blue)
 - Manager
 - Operator
 - NewLevel
 - Weekend Access
 - Roles
 - Administrator
 - Operator
 - Weekend Security

On the right, a modal dialog titled "Rules" is open. It contains two fields:

- Level Name: Intern
- Rule: containsAny([security-zones], 'PlantA', 'Floor1', 'Press Room')

A red box highlights the "Rule" field.

4. After you enter your rule, click **Save**.

Troubleshooting Identity Providers

This Troubleshooting section has a compilation of examples to help you diagnose and troubleshoot issues with configuring IdPs.

The Save Button Is not Selectable

All required fields must be entered on the Identity Providers screen before Save can be selected. Required fields have an asterisk (*) next to their name.

Note:

If you are waiting for values for the ClientID and Client Secret fields, you can enter fake values and return when you have the correct value.

After Importing Metadata from a File, Values Did not Auto Populate

1. Verify that import file is JSON format.
2. Verify that the import URL is valid.
3. Re-import the file.

Login Testing: IdP Login Page Does not Appear

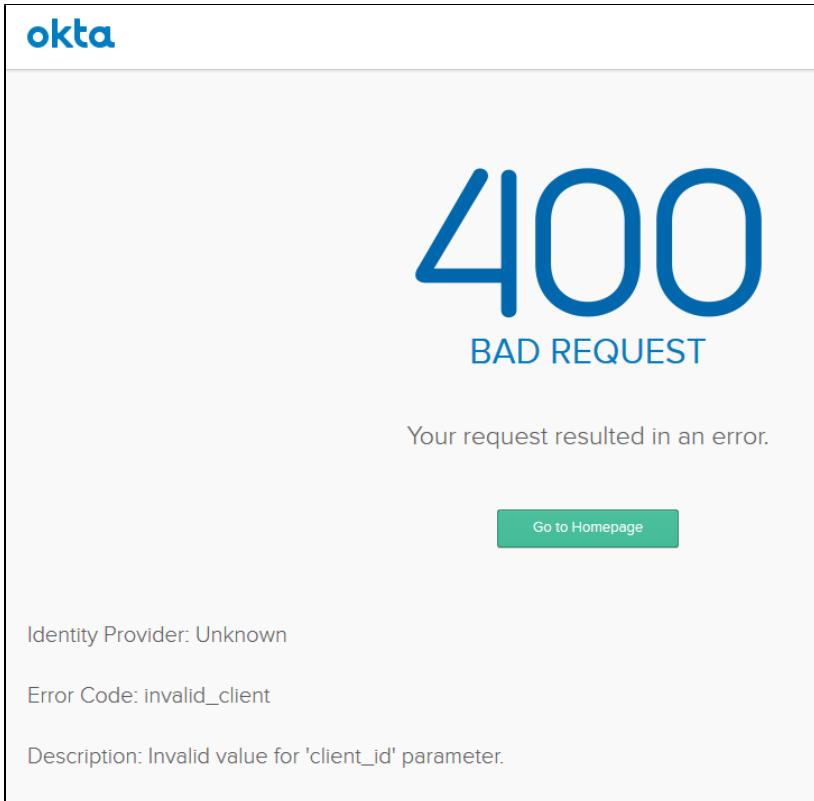
1. Confirm that the values in **Client ID** and **Client Secret** fields are correct. These come from the IdP when your Ignition Gateway is added as an application.
2. Check all other configuration settings.

On this page ...

- [The Save Button Is not Selectable](#)
- [After Importing Metadata from a File, Values Did not Auto Populate](#)
- [Login Testing: IdP Login Page Does not Appear](#)
- [Login Testing: The IdP login Is Displayed but the Login Attempt Fails](#)
- [Login Testing: The IdP Login Accepts the User but the IDP Redirect Fails \(HTTP ERROR 500\)](#)
- [Login Testing: The Test Is Successful, but Results Do not Show Useful data \(i.e., user name, eMail\)](#)
- [Login Testing: Revised User Attributes Are not Shown in the Results of a Successful Test](#)
- [You Are not Re-directed Back to Ignition after a Successful IdP Login](#)

Login Testing: The IdP login Is Displayed but the Login Attempt Fails

This issue is outside of Ignition. Check with your IT department and verify the login credentials (username and password) for your IdP.



Login Testing: The IdP Login Accepts the User but the IDP Redirect Fails (HTTP ERROR 500)

1. Go to Settings for your IdP.

Name	Type	Description	Action
Default_User_Source	Ignition		More Delete
Okta_2018	OpenID Connect 1.0		More Delete
Okta_Example	OpenID Connect 1.0		Settings More Delete
Okta_TroubleShooting	OpenID Connect 1.0	User Attribute Mapping User Grants	More Delete

2. Verify the setting for **Supported ID Token Signing Algorithm Values**.
3. If the URL for the IdP's metadata is available, try re-importing it.
4. Verify and re-enter the Client Id and Client Secret.
5. Verify the Token URL. Then re-import the IdP's metadata.
6. Verify the JSON web keys URL. (Default is to leave the check box checked.) Then re-import the IdP's metadata.
7. Verify the Issuer URL, then re-import the IdP's metadata.

HTTP ERROR 500

Problem accessing /data/federate/callback/oidc. Reason:

com.inductiveautomation.ignition.gateway.auth.idp.IdpAdapterManagerException: Unable to parse attributes from the web auth response

Login Testing: The Test Is Successful, but Results Do not Show Useful Data (i.e., user name, eMail)

1. Go to settings for your IdP.
2. Add the desired fields to the Scope section. You may have to reference the developer documentation Scope document

Scope	Name	Action
	email	Remove
	user name	Remove

3. Click Save.
4. Repeat the Login test.

Login Testing: Revised User Attributes Are not Shown in the Results of a Successful Test

1. Go to settings for your IdP.
2. Add the desired fields to the Scope section.
3. Click Save.
4. Repeat the Login test.

You Are not Re-directed Back to Ignition after a Successful IdP Login

1. Verify the Adapter Configuration: Authorization URL.
2. Re-import the IdP's metadata.

Referencing User Information

When we work with Identity Providers, the user signs into the provider, and the provider sends Ignition a response document containing information about the user that successfully logged in. This document is the basis for how Ignition handles the user, including what ID, username, roles, and contact info the user is assigned. When configuring a custom identity provider in Ignition, it will often be necessary to determine the contents of the response document, and make decisions about the logged-in user based on what is found.

Here, we'll discuss some of the ways of referencing user information in Identity Provider-driven expressions.

On this page ...

- [Accessing a Response Document](#)
- [Referencing Response Elements](#)
 - [Reference Structure](#)
 - [Special Object Reference](#)
 - [Special Function Reference](#)
 - [Evaluating Tag Values](#)
 - [Special Considerations for Rules](#)
- [Referencing Mapped Attributes](#)

Accessing a Response Document

Many of the expressions found in the following sections will require us to know the layout of our Identity Provider's response document, as returned for a user when they authenticate. To view the structure of the response document for a given user, the [Test Login](#) feature can be used for the provider. The **IdP Response Data** tab will contain the returned document.

The screenshot shows the Ignition Test Results interface with the 'Identity Providers' section selected. The 'IdP Response Data' tab is active, displaying a JSON response document. The JSON structure includes a token endpoint response object and an id token claims object. The token endpoint response contains an access token, id token, token type (Bearer), expires in (3600), and scope (openid). The id token claims object contains iss (default), aud (ignition), exp (1602202511), jti (pB7xNnbS93vARr_ijgygTQ), and iat (1602201911).

```
{
  "tokenEndpointResponse": {
    "access_token": "pE7Z8qg-06EJIrLyi59ceheFHljEZFnM4FEx5dBoieY",
    "id_token": "eyJraWQiOiJrMSIsImFsZyI6IlJTMyU2In0.eyJpc3MiOiJkZWZhhdWx0IiwiYXVkJjoi
    "token_type": "Bearer",
    "expires_in": 3600,
    "scope": "openid"
  },
  "idTokenClaims": {
    "iss": "default",
    "aud": "ignition",
    "exp": 1602202511,
    "jti": "pB7xNnbS93vARr_ijgygTQ",
    "iat": 1602201911,
    "nbf": 1602201701
  }
}
```

Referencing Response Elements

Whenever we work with a response document in an Ignition expression, as from [User Attribute Mappings](#) or [Security Level Rules](#), we may need to reference items in the document directly.

Reference Structure

In IdP contexts, it is possible to reference IdP document elements using a three-part format: {attribute-source:X:Y}

In this format, "X" is the attribute source for the provider, while "Y" is a path to the property inside the attribute source. Possible values for the attribute source will vary by IdP type:

IdP Type	Attribute Source(s)	Expression Path Name
Ignition	ID Token Claims (default)	idTokenClaims
	Token Endpoint Response	tokenEndpointResponse
	User Info Claims	userInfo
OIDC	ID Token Claims (default)	idTokenClaims
	Token Endpoint Response	tokenEndpointResponse
SAML	Authentication Response	authnResponse

In Ignition and OIDC Identity Providers, the Attribute Source is included in the response document already, with member properties inside. For SAML Identity Providers, the authnResponse string will not be shown in the structure, but must be included in the expression.

Special Object Reference

These special objects can be used to reference information gathered from the IdP response document or the Security Zone that the user falls under.

Object Name	Description
{security-zones}	This object gives the collection of security zones that the user currently has. The collection can be handled using one of the unique functions: containsAll or containsAny.
{attribute-source:X:Y}	This object can be used to access values in deeper structures by using colons to delimit each object. Where "X" is the attribute source for the provider, as defined on the User Attribute Mapping page. "Y" is any number of additional attributes along the path to the desired attribute.

Assuming the following JSON response from an identity provider:

```
{
  "idTokenClaims": {
    "foo" : "bar"
  },
  "userInfo": {
    "email" : "person@place.com"
  }
}
```

{attribute-source:userInfo:email} would retrieve the value of the user's email.

Attribute	Path	Example
ID	sub	{attribute-source:idTokenClaims:sub}
Username	preferred_username	

		{attribute-source:idTokenClaims:preferred_username}
First Name	given_name	{attribute-source:idTokenClaims:given_name}
Last Name	family_name	{attribute-source:idTokenClaims:family_name}
Email	email	{attribute-source:idTokenClaims:email}
Roles	roles	{attribute-source:idTokenClaims:roles}

Note: The {attribute-source:x:y} object is only available to Security Levels that fall within the Authenticated Security Level. See below under Special Considerations for Rules.

Special Function Reference

When writing an expression to determine Security Level Rules, there are functions available that are not a part of the normal set available to Expression Bindings. These additional functions are:

Function Name	Description	Example
containsAll(collection, element 0, ..., element N)	Checks to see if all of the listed elements are present in the collection object. The function requires at least two arguments, a collection and an element.	<pre>containsAll({security- zones}, 'PlantA', 'Floor1', 'Press Room')</pre>
containsAny(collection, element 0, ..., element N)	Checks to see if any of the listed elements are present in the collection object. The function requires at least two arguments, a collection and an element.	<pre>containsAny({attribute- source:idTokenClaims: roles}, 'Manager', 'Operator')</pre>

Evaluating Tag Values

Tag values can be accessed in the Security Level Rules expression area by encasing the Tag Path (including the Tag Provider) in braces ("{}")

```
{[tagProvider]path/to/tag}
```

The screenshot shows the 'Config > Security > Identity Providers > default - Security Level Rules' interface. On the left, a tree view shows 'Public' expanded, with 'Authenticated' and 'CustomLevel' as children. 'CustomLevel' is selected and highlighted in blue. On the right, under the 'Rules' section, there is a 'Level Name' field containing 'CustomLevel' and a 'Rule' field containing the expression '[[Sample_Tags]Writeable/WriteableBoolean2]'. A vertical scroll bar is visible on the right side of the main content area.

Note: Security Levels are determined on initial login for each session, so if a Security Level is using an expression that references a Tag value, changing the value while the session is running won't change the Security Levels applied to the users already logged in.

Special Considerations for Rules

When defining rules for a Security Level, it is important to notice where in the Security Level tree you are. If you want to access information out of the Identity Provider such as the username, you will need to ensure that the Security Level is located in the Authenticated branch. User information is only captured once a user logs in, so that information will only fall under Security Levels that come from the user being Authenticated. If a Security Level lies outside of the Authenticated branch, then the level will only have access to information such as Tag values and Security Zones.

Referencing Mapped Attributes

In the context of [Security Level Rules](#), it is possible to reference response document elements as discussed above, but it is also possible to use mapped attributes directly.

Service Security

Service Security

After creating some Security Zones, a Security Policy can then be defined for each zone. This can be found by going to the **Config** section of the Gateway Webpage and navigating to **Security > Service Security**. At first, none of the zones will have a policy defined, and the Default zone will be at the top. Editing any of them will bring up the Security Policy definition page for that zone. The Security Policy has four sections: Alarm Notification, Alarm Status, History Provider Access, and Tag Access. They work together to define how the local Gateway gives access to incoming Gateway connections. All four sections also have the ability to completely block access to specific services with the Service Access setting in each section. Setting that to deny will deny the zone access to that particular information, regardless of what the rest of the options are set to.

Note: It is important to realize that if you have a single Gateway, limiting access of certain clients to certain Tags is still done in the [individual Tags](#).

This feature is new in Ignition version **8.0.7**

[Click here](#) to check out the other new features

On this page ...

- [Service Security](#)
 - [Default Security Zone](#)
 - [Setting Zone Priority](#)

- **Alarm Journal Access** - Alarm Journal Access has two main settings associated with it. The Default Profile Access is the default access rights for the Alarm Journal service. There will be an additional setting for each Alarm Journal configured on the local Gateway. As an example, the image below shows an "Access Level: 'MyJournal'" setting which corresponds to the configured Alarm Journal named "MyJournal". This setting can be set to **Inherited** which will cause this specific Alarm Journal to inherit the access rights set in the Default Provider Access Level. It can also be set to **No Access** to block query and storage to this specific Alarm Journal. Setting it to **Query Only** will allow users to only query data from this Alarm Journal without any storing capability. In contrast, the **Query and Storage** option allows users to store and query data from this Alarm Journal. It is important to note that every time a new Alarm Journal is created in the local Gateway, a new setting for this journal will be added to this Security Policy and it will automatically default to inherited.

- **Alarm Notification** - The Accessible Pipeline Filter setting is a list of Pipelines in the current Gateway that other connections can use for alarm notification. Pipelines must be entered in the format "project_name/pipeline_name". The list is a comma separated list, and it can make use of the (*) wildcard. This setting is an inclusionary list not an exclusionary list, meaning that if there are no pipelines listed here, then all of them will be available.

- **Alarm Status** - The Allow Acknowledge setting will allow the Gateways that fall within the zone to acknowledge alarms on the local Gateway. The Allow Shelving setting will allow the Gateways that fall within the zone to Shelve alarms on the local Gateway. IE: Other Gateways can shelve alarms on this Gateway. For this Gateway to shelve alarms on others, this must be set on the remote Gateway.

This feature is new in Ignition version **8.0.7**

[Click here](#) to check out the other new features

- **Audit Log Access** - Audit Log Access has two main settings associated with it. The Default Profile Access is the default access rights for the Audit Profile service. There will be an additional setting for each Audit Profile configured on the local Gateway. Similar to the Alarm Journal Access, the image below will show an "Access Level: 'MyProfile'" setting which in this case corresponds to the configured Audit Profile named "MyProfile". This setting can be set to **Inherited** which will cause this specific Audit Profile to inherit the access rights set in the Default Provider Access Level. It can also be set to **No Access** to block query and storage to this specific Audit Profile. Setting it to **Query Only** will allow users to only query data from this Audit Profile without any storage ability. In contrast, the **Query and Storage** option allows users to store and query data from this Audit Profile. Just like with Alarm Journals, it is important to note that every time a new Audit Profile is created in the local Gateway, a new setting for this profile will be added to this Security Policy and it will automatically default to inherited.

- **History Provider Access** - The History Provider Access has two different settings. First, it has a Default Access Profile. This is the default access rights for Tag History. Second, there will be a setting for each History Provider setup on the local Gateway. In the image below, there is an "Access Level: 'DB'" that can be set that corresponds to the History Provider that was created when a database was connected. It can be set to Query and Storage, which will allow connections in the current zone to both run queries and store Tag History against the Tag History provider, Query Only, which will only allow the zone to query out history data, but not store it, and No Access, which will completely block access to that History Provider. The final setting is Inherited, which will inherit the Default Profile Access rights. Any new history providers will automatically get added to the Security Policy set at inherited so it may be

beneficial to set the Default Profile Access to be either Read Only or No Access so that a recently added history provider does not accidentally get storage rights when it should not.

Note: The Default Access Profile should not be set to Inherited. This also goes for the Default Provider Access Level in the Tag Access section.

- **Tag Access** - The [Tag Access](#) section also has a few different settings. The Default Provider Access Level sets the default access rights for realtime Tag providers.

The Impersonation Role Name field allows you to specify a role name to use when writing to a Tag from an incoming Gateway Network connection (from the selected Zone). Finally, the Tag Access section will then have a setting for each Tag provider configured in the local Gateway, as well as an additional one for system Tags. These can be set to **ReadWriteEdit**, which will allow connections in the current zone to read, write to, and edit the Tags in that provider, **ReadWrite**, which allows the zone to read and write to Tags, and **ReadOnly**, which only allows the zone to read the Tags. It also can be set to **None**, which will prevent the zone from interacting with the Tag Provider altogether, and **Inherited**, which will again inherit the access rights set in the Default Provider Access Level. Any new Tag Providers will automatically get added to the Security Policy with Inherited access rights.

This feature is new in Ignition version **8.1.2**
[Click here](#) to check out the other new features

As of 8.1.2, the Trust Remote Security Levels setting allows users to opt into trusting the Security Levels of remote Gateway users when remote Gateways read, write, and subscribe to local Tags. If checked, security levels passed from the remote Gateway will be used for determining access to Tags on the local Gateway. If unchecked, or if the remote Gateway is on a version which does not support this feature, the remote Gateway's security zones and the impersonation role will be used as the security levels.

Default Security Zone

While the Default zone may not have a custom Security Policy defined, it does default to not include any notification pipelines, allow alarm acknowledgment, query only history access, and read only Tag access. This means that if a remote Tag Provider is setup on a remote Gateway, and the local Gateway has not changed the default security settings, the remote Gateway will have read only access to the Tag History Provider. This can be changed by editing the Default zone's Security Policy to fit a different preference, or creating new Security Zones with custom security policies. Once a Security Policy has been defined on a zone, it will automatically jump to the top of the list. A new option will also become available that will clear the policy from the zone.

Alarm Journal Access	
Service Access	Allow
Default Profile Access	Query Only
Access Level: 'MyJournal'	Inherited

Alarm Notification	
Service Access	Allow
Accessible Pipeline Filter	A comma separated list of alarm pipeline names on this gateway (which can include the wildcard "") that will be made available for use in the alarm notifications of other gateways in this security zone.

Alarm Status	
Service Access	Allow
Allow Acknowledge	<input checked="" type="checkbox"/>
Allow Shelving	<input type="checkbox"/>

Audit Log Access	
Service Access	Allow
Default Profile Access	Query Only
Access Level: 'MyProfile'	Inherited

History Provider Access	
Service Access	Allow
Default Profile Access	Query Only
Access Level: 'DB'	Inherited

Tag Access	
Service Access	Allow
Default Provider Access Level	ReadOnly
Trust Remote Security Levels	<input type="checkbox"/> If checked, security levels passed from the remote Gateway will be used for determining access to tags on the local Gateway. If unchecked, or if the remote Gateway is on a version which does not support this feature, the remote Gateway's security zones and the impersonation role will be used as the security levels.
Impersonation Role Name	<input type="text"/> This role name will be injected into the security model of the tag for read/write/subscribe/edit operations.
Access Level: 'default'	Inherited
Access Level: 'System'	Inherited

Setting Zone Priority

Once a Security Policy has been defined for two or more zones, a new option appears on the Service Security page to move the zones up and down the list. This allows a priority to be set on the Security Zones, since a connection can apply to multiple zones. For example, say Zone 2 dictates that all requests coming from a range of IP addresses have query only history access, and read write access to Tags. Zone 1 includes specific Gateways, one of which is also contained in Zone 2, that will have query and storage history access and read write edit access to Tags. When a request comes in from a connection, it first determines which Security Zones it belongs to. The request then starts at the top of the Service Security list and goes down until it finds the first zone that it is in, and uses the access rights of that zone. In our example, we want to make sure Zone 1 is above Zone 2, so that the Gateway that is in both Zone 1 and Zone 2 gets the full access rights afforded to it by the Security Policy of Zone 1 instead of getting the limited access rights from Zone 2.

Service Security

⚠ Security policies are defined based on Security Zones. The highest ranking policy (from the top down) for a connection's zones will be used. If no other policies match, the "Default" policy will be used.

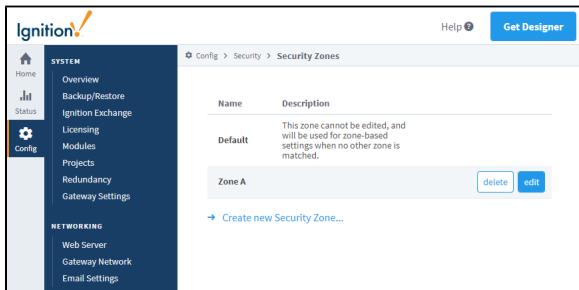
Security Zone	Policy Defined?	
Zone 1	true	Edit Clear Policy Move Down
Zone 2	true	Edit Clear Policy Move Up
Default	false	Edit

Related Topics ...

- [Security in Vision](#)

Security Zones

A Security Zone is a list of Gateways, Computers, or IP addresses that are defined and grouped together. This group now becomes a zone on the [Gateway Network](#), which can have additional policies and restrictions placed on it. While Users and Roles restrict access to specific functions within the Gateway like making certain controls read-only for certain users and read/write for others, Security Zones provide this functionality to the Gateway Network, limiting locations instead of people to be read-only for specific actions. This allows for greater control over the type of information that is passing over the network, improving security and helping to keep different areas of the business separate, while still allowing them to interconnect.



Using Security Zones

Sometimes, in addition to knowing who the user is, it is important to know their location. An operator may have permissions to turn on a machine from an HMI, but if the operator is logged into a project on a different Gateway in the network that has remote access to those Tags, it might not be a good idea to let the operator write to those Tags from a remote location. The operator can't see if the physical machine is clear to run.

This is where Security Zones come in. While Security Zones themselves don't define the security, they instead define an area of the Gateway Network, breaking up Gateways and network locations into manageable zones that can then have a Security Policy set on them. Once there are zones defined, a Security Policy can be assigned to each zone, and a priority of zones can be set in the event that more than one zone applies in a given situation.

Caution: When using zone-based security in a project, the project stores the name of the security zone as a string. This means that if you were to modify the name of the zone in the Gateway, the zone-based security in your project will not update to reflect the new name, and instead will try searching for a zone with the original name. Be very careful when modifying the names of security zones.

A connection must pass all of the qualifier checks before being accepted into a Security Zone. So if Require Secure Connection was checked, and Allow Client Scope was not, any requests coming from Clients would be rejected even if they are secure, and the same goes for any non-secure connections coming from sources other than a Client.

Requests can be a part of more than one zone, depending on how the zones are setup. This can be useful for making a whole section of IP addresses read only, but a specific Gateway in that IP address range may be listed specifically in another zone, which can be given read/write access. Any connection which does not fall into one of the zones will be placed in the Default zone.

Define a Security Zone

When setting up a new Security Zone, it is a good idea to setup a [Gateway Network](#) first if you haven't already. While Security Zones can be defined and used without a connected Gateway, they work best when used in conjunction with other Gateways on a Gateway Network.

Security Zones are defined

There is a special zone called Default. It is always present and can't be modified, and will be used if an incoming connection does not match any of the other defined zones.

1. Under the **Config** tab of the Gateway Webpage go to **Security > Security Zones**.
2. Select the **Create new Security Zone** link.

On this page ...

- [Using Security Zones](#)
- [Define a Security Zone](#)
- [Settings Table](#)



Security Zones and Service Security

[Watch the Video](#)

3. Enter a name and description for the new zone.
4. The identifiers are how incoming connections are distinguished between different zones. While there are a few different ways to define the incoming connection, it only needs to match one of them to match this zone. In the Identifiers section, enter an IP Address and a Gateway name.
After first being identified as part of a particular Security Zone, the connection then must check the Qualifiers. With the Qualifiers, the incoming connection needs to fit in with all of the properties before it is fully placed into the Security Zone. In the Qualifiers section, select the Require Secure Connection option. Leave the others at their default.s
5. Click **Create New Security Zone**. The page will refresh and you will see a green banner stating that your new Security Zone was successfully created.

Settings Table

Setting	Description
Identifiers	
IP Addresses	This defines an IP address that the connection is coming from. This can be a list of IP addresses by using commas to separate them. It can also make use of the (*) wildcard like '192.168.100.*', or use a range such as '100.100.1-100.0-255'. With IP addresses, virtually all connections can be listed. Use 127.0.0.1 for the local connection.
Host Names	The host name refers to the system name of the machine generating the request such as Joe_Workstation. This can be a list of names separated by commas, and it can also use the (*) wildcard like '*_Workstation'.
Gateway Names	The Gateway name is the name of the Ignition instance. This is set in the Config section by going to System > Gateway Settings , and changing the System Name. This can be a comma separated list and can use the wildcard such as '* Ignition Gateway'. <p>Note: When identifying a Gateway through a proxy Gateway, the IP Address should be using the IP of the proxy, but the Gateway name should use the name of the Gateway we are trying to identify.</p>
Qualifiers	
Require Secure Connection	If this is true, only connections that are made over a secure channel will be accepted.
Direct Connection Required	If this is true, only connections that come from a direct connection will be accepted. The Gateway Network allows you to connect three Gateways in a 1-2-3 configuration, where Gateway 1 can see Gateway 3 through the proxy Gateway 2.
Allow Client Scope	If this is false, any client scoped requests will not be accepted.
Allow Designer Scope	If this is false, any Designer scoped requests will not be accepted.
Allow	If this is false, any Gateway scoped requests will not be accepted.

Related Topics ...

- [Service Security](#)

Project Security in the Designer

When several users are all working on the same project, managing changes to the project can become cumbersome. By default, all users with Designer access can modify, delete, save, and publish all resources available in the Designer. In some situations, it is desirable to limit what each user can do in the Designer. Ignition has several built-in Designer restriction methods to help in these scenarios.

On this page ...

- [Designer Project Permissions](#)
 - [Controlling Project Edits by Role](#)
 - [Protecting Project Resources](#)

Designer Project Permissions

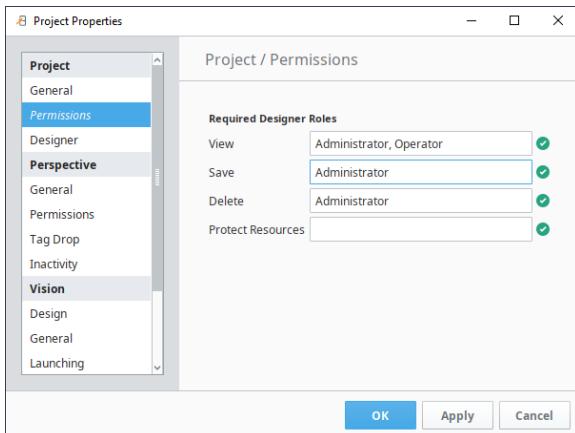
Actions such as publishing, viewing, saving, deleting, and editing of project resources are restricted to users who have sufficient roles to do so. Editing of the these required roles is done in the permissions section of the Project Properties dialog in the Designer. If required roles are not set for an action, then all users with Designer access can perform the action.

The Designer does not poll for role changes, so if a user who is currently logged into the Designer has their roles changed, they will need to re-launch the Designer for the new role(s) to take effect.

Controlling Project Edits by Role

You can control who gets to login to a project by assigning roles and giving permissions to those roles in the Required Designer Roles property which you set up in the Designer.

1. In the Designer, from the menubar, choose **Project > Properties**.
2. Go to the **Project > Permissions** area.
3. Under the Required Designer Roles, enter the appropriate roles next to each project-level restriction, as required. You can enter a comma-separated list of role names that are required to access the project. As you start typing, matching role names will pop up.



4. Click **OK** to save the changes.

The following table describes each of these five options:

Option	Affect
View	User must have at least one of these roles to view the project in the Designer.
Save	User must have at least one of these roles to save the project.
Delete	User must have at least one of these roles to delete the project.
Protect Resources	User must have at least one of these roles to access protected resources.

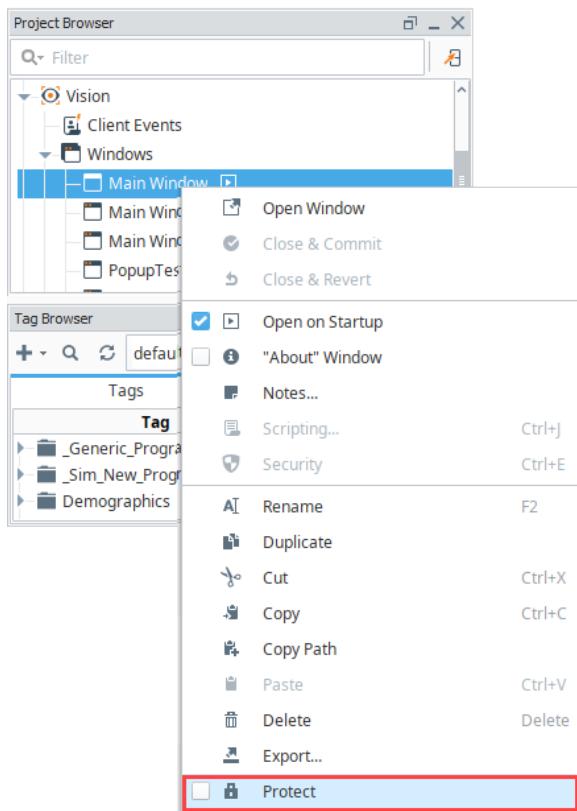


Project Permissions

[Watch the Video](#)

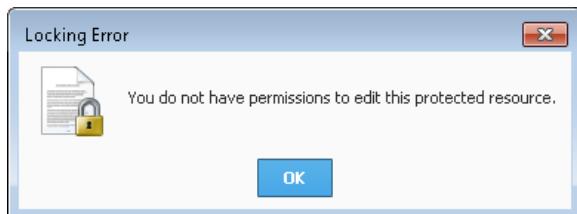
Protecting Project Resources

You can lock individual project resources from inside Designer by opening the Project Browser, and right clicking on any of the objects that you want to lock in. Select the **Protect** option to protect it. Once it's protected, it cannot be changed except by someone that has the permission to unprotect it, and modify it.

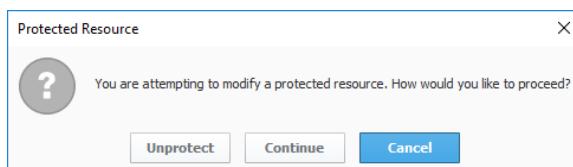


Protected resources are global or project resources that can only be edited by select users with the required roles. These roles are required to protect resources from being edited in the Designer, and do not apply to the clients. This means you can prevent a resource from being edited by other users who have Designer access. It is often used in scenarios where development work is finished on a window or object, and no further changes should be made to it. Other objects like [Vision Templates](#) or [Alarm Pipelines](#) are often protected so they may be used, but not modified.

Users without a required role will see the following message in the Designer when attempting to open a protected resource:



Users with a required role are allowed to modify the resource, but a message will appear informing them that the resource is protected, and will be asked how to proceed:

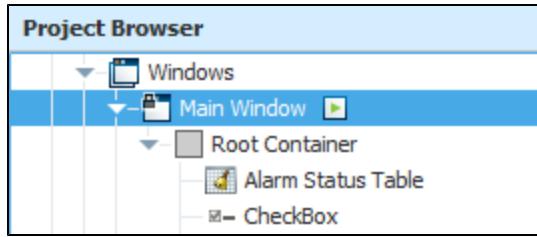


Additionally, a lock icon will appear on the resource informing users that it is protected. An example can be seen on the 'Audit Events' window below:



Locking Project Resources

[Watch the Video](#)



To remove the protection, simply right click the object and select the **Protect** option to unprotect it.

Designer

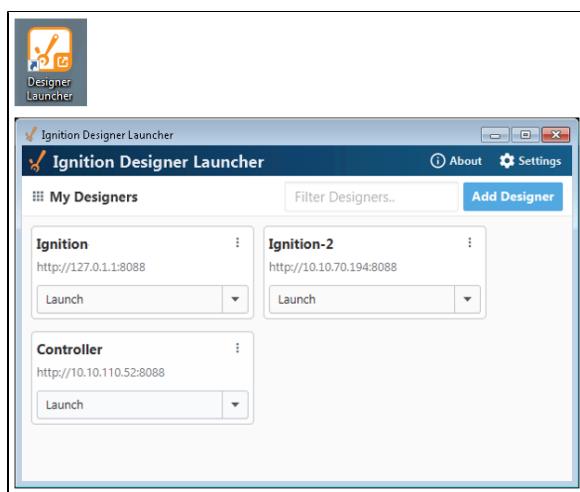
The Ignition Designer is where the majority of configuration and design work is done. The Designer uses web-launch technology to open and edit your projects. This is how you can configure your [Perspective](#) and [Vision](#) projects, and much more.

The Designer provides all the firepower to bring your projects to life. It uses a drag-and-drop configuration making screen development quick and intuitive. You can create user interfaces by dragging [Perspective components](#) onto a view, [Vision components](#) onto a window, and Tags onto your components to instantly bind data to tables, charts, graphs, and more. You can set up Tags and [Transactions](#) to log data to your databases. You can set up [Reports](#) to generate and save data however you'd like. The Designer saves all your projects to the Gateway so everything is controlled in one place.

This page provides some good information about the Designer, Designer workspace, Tools, and how to create a project.

Opening the Designer

Opening the Designer requires running the [Designer Launcher](#). Once the Designer Launcher is installed and configured, you'll have all your projects at your fingertips. If you created a desktop shortcut for the Designer Launcher at install, simply click to open it and select a Designer to open. From there you can create a new project or open an existing project.

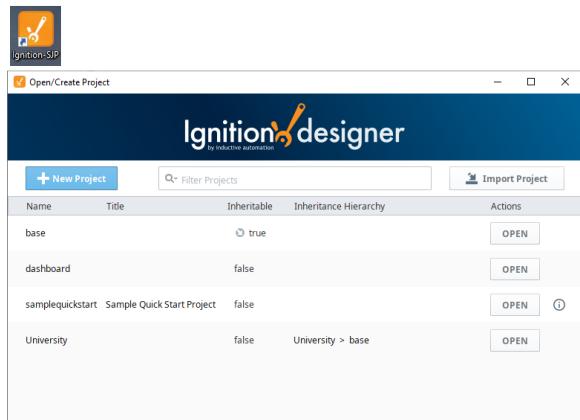


On this page ...

- [Opening the Designer](#)
- [Creating a Project](#)
 - [Project Creation Settings](#)
 - [Open or Create a Project from within the Designer](#)
 - [Updating Project Settings](#)
- [Designer's Workspace](#)
- [Designer Tools](#)
 - [Tools Menu](#)
 - [Previewing the Project](#)
 - [Find and Replace](#)
 - [Keyboard Shortcuts](#)
- [Concurrent Editing and Conflict Resolution](#)
 - [Concurrent Users UI](#)
 - [Conflict Resolution](#)

The image features the Inductive University logo, which consists of a blue hexagon with the letters "IU" and a green laurel wreath. To the right of the logo, the text "INDUCTIVE UNIVERSITY" is written in a serif font. Below the logo, the text "The Designer Launcher" is displayed in bold, followed by a link "Watch the Video" in blue.

If you have a Designer that has projects that you work on regularly, you can create a shortcut to that Designer and keep it on your desktop. When you click on your shortcut, it opens the Designer, then click on the project you want to edit. Now you can hit the ground running designing your project!

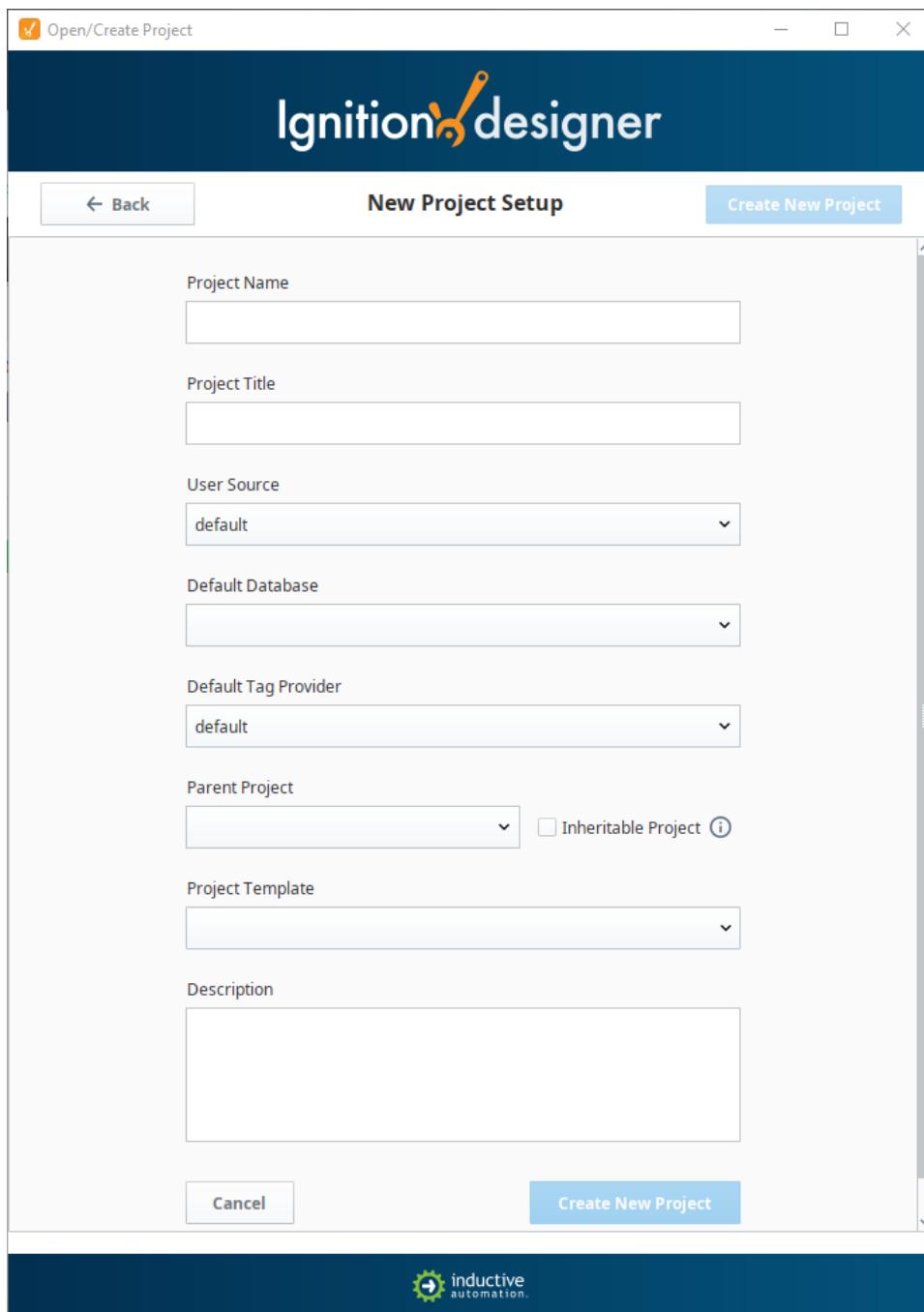


Refer to the [Designer Launcher](#) page for downloading, installing, and configuring the Designer Launcher.

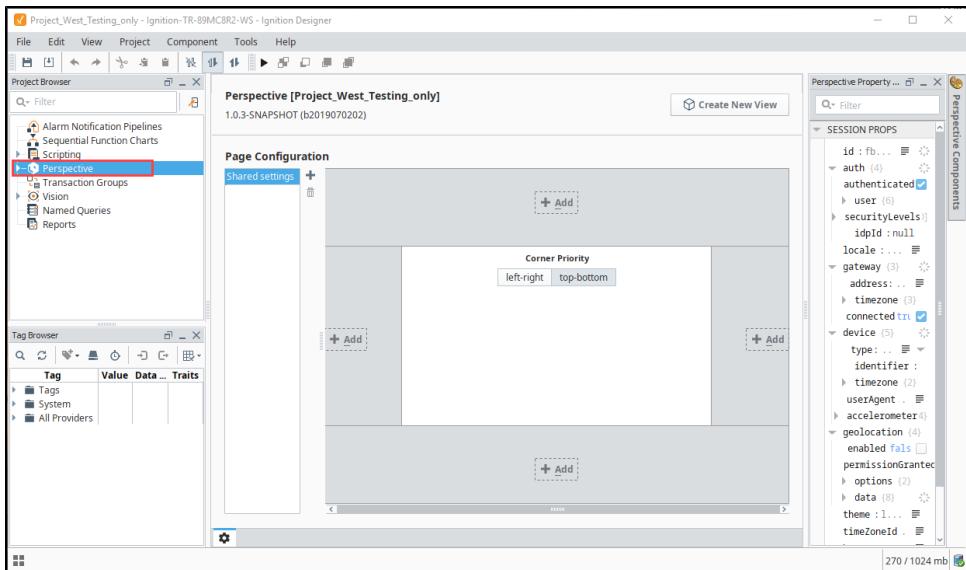
Creating a Project

The first step in working with a project is creating one. Once you launch the Designer, the Open/Create Project window is displayed. Here you have the option to create a new project or open an existing project. Let's create a new project!

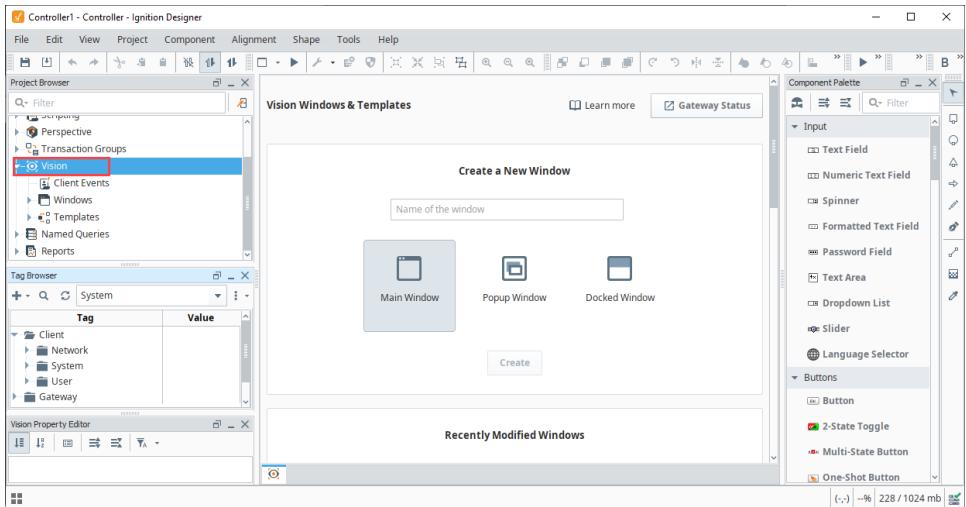
1. After opening the Designer, click on **Create a New Project** button. The New Project Setup window is displayed.



2. Enter the **Project Name** (required) and any other configuration settings you need for your project. Most settings are optional. Refer to the [New Project Creation Settings](#) Table below for a description of each property.
3. Click on **Create New Project**.
4. Your project is now created and opens on the Perspective Configuration Page. If you want to design your project in Vision, click on the **Vision** application in the Project Browser and the **Vision Getting Started** window will open.



To design your project in Vision, expand **Vision** in the Project Browser to create windows, add components, Tags, and more.



Project Creation Settings

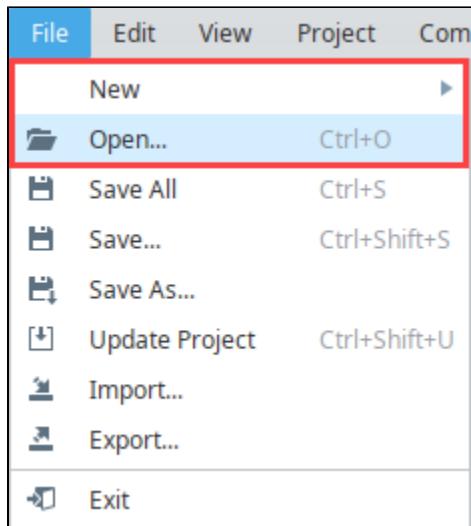
The table below contains descriptions for available settings when creating a new project in the Designer.

Property	Description
Project Name	<p>The Project Name can only consist of alphanumeric characters and the '_' (underscore) character. Spaces and other special characters are not supported.</p> <p>Note: It is not advisable to change the Project Name after it's been created, instead, change the Title property.</p>
Project Title	This is the name that will be displayed on the launch page of the Gateway and in the runtime Client or Session, (optional) There are no restrictions on using special characters or spaces. If you do not specify a title, the project name will be displayed on the Gateway launch page and in the runtime.
User Source	Determines the User Source associated with this project (when using the Classic authentication strategy).
Identity Provider	Determines the Identity Provider associated with this project (when using the Identity Provider authentication strategy).
Default Database	Select a Default Database (optional). Any queries to the database will use this database connection unless explicitly specified otherwise.

Default Tag Provider	Select a default Tag Provider (optional). If left blank, bindings and references to tags will always need to include the tag provider
Parent Project	Each project may have a parent project , and will inherit all of the resources of that parent project, (optional). Click the dropdown list to see all the available options.
Project Template	Select a Project Template (optional). There are several pre-built project templates focused on navigation that support either Perspective or Vision. Click the dropdown to see all the available options.
Description	Enter a description of the project (optional). Once a project is created, this description can be viewed on the Open/Create Project screen when you hover over the Information  icon.

Open or Create a Project from within the Designer

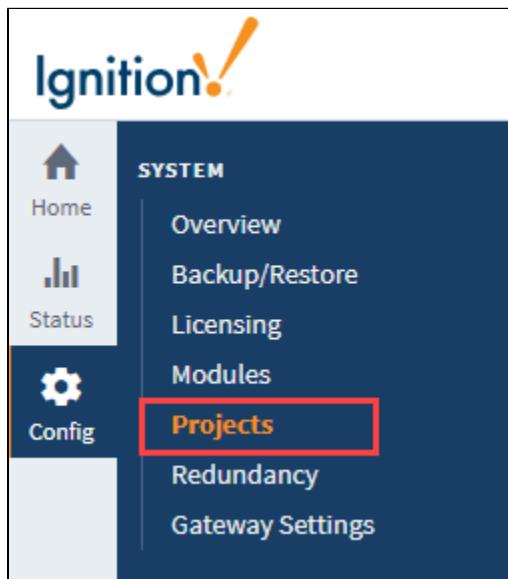
To create a new project or open a different project from within the Designer, use the **File > Open** menu in the top menubar.



The Open/Create Project screen will be displayed. You can choose from existing projects or create a new project.

Updating Project Settings

Project settings such as the title, description, connections, and inheritance are set through the Gateway Webpage **Config Tab**, under **System > Projects**. For more information, see [Projects](#).

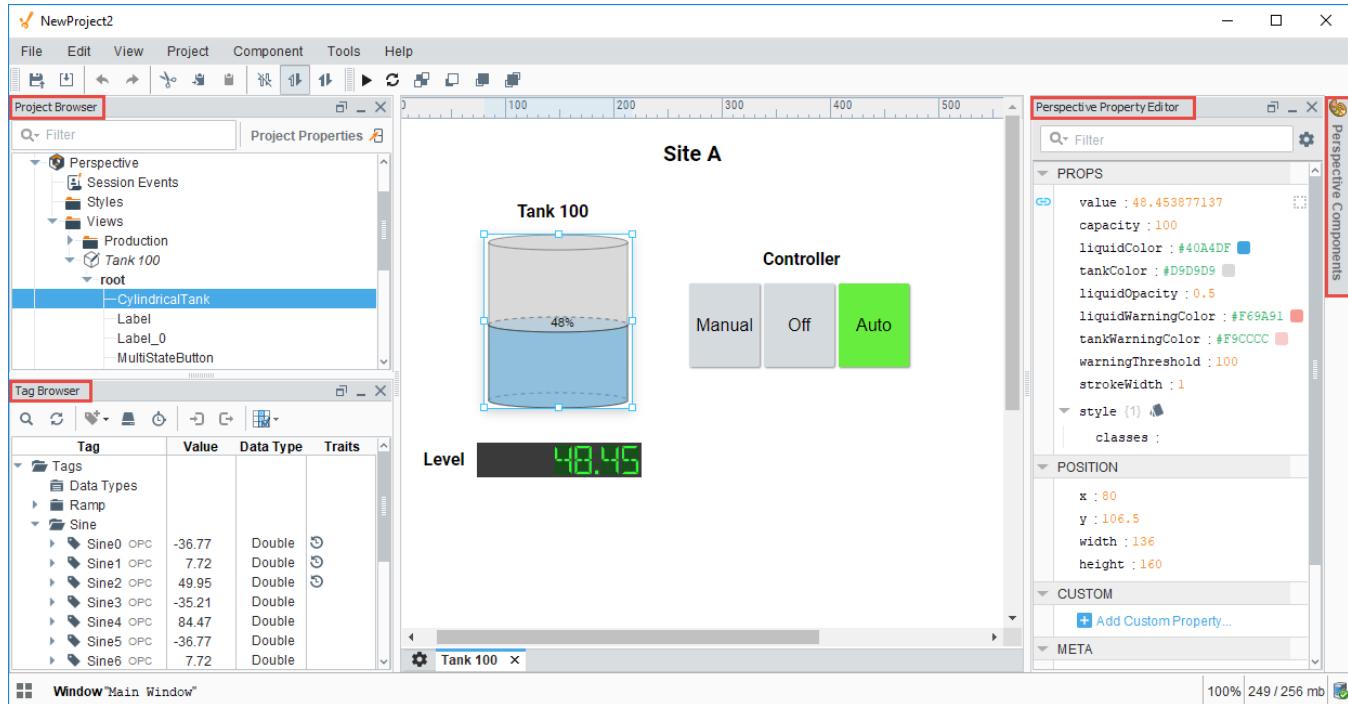


Designer's Workspace

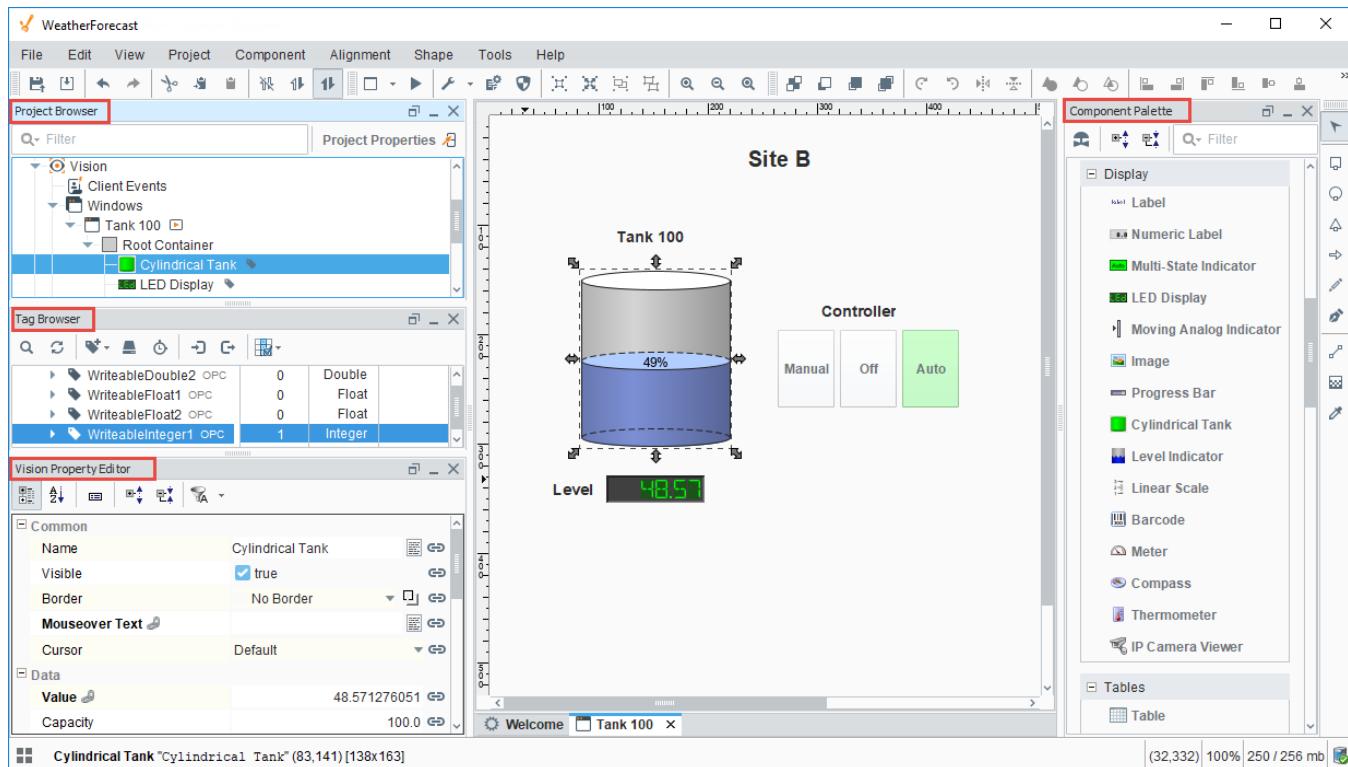
The Designer workspace is centrally located and organized by panels. Some of the panels include a Project Browser, Tag Browser, Component Palette, and Property Editor. These panels can change depending on the type of resource you are currently editing. For example, if you are editing a Perspective view or Vision window, the Designer workspace has Component Palette and Property Editor panels. If you're editing an Alarm Notification Pipeline, your Designer workspace will be the Pipeline Block Editor. If you're editing a Report, your Designer workspace will be the Report Designer. Each type of workspace has panels that are only valid when that workspace is active.

Here are two images showing the Perspective Designer workspace and Vision Designer workspace, and an example displaying the same components. At a glance, they look very similar, but there are some differences, including each having their own Component Palettes. To learn more, refer to the [Perspective Designer Interface](#) and the [Vision Designer Interface](#) pages.

Perspective Designer's Workspace



Vision Designer's Workspace



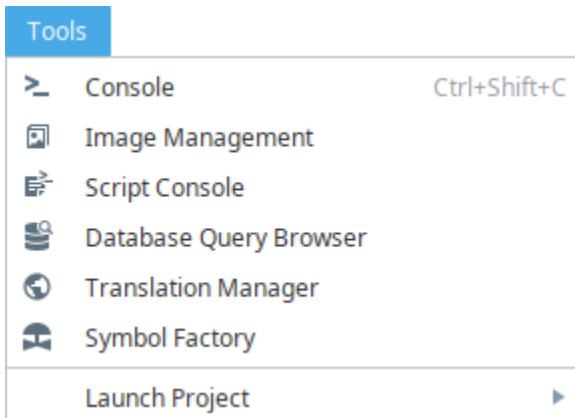
Designer Tools

The Designer has a host of tools to help you accelerate building, testing, and deploying your project. Let's talk about a few here.

Tools Menu

In addition to all the panels available in the Designer workspace, there is also a [Tools menu](#) to help you create your projects.

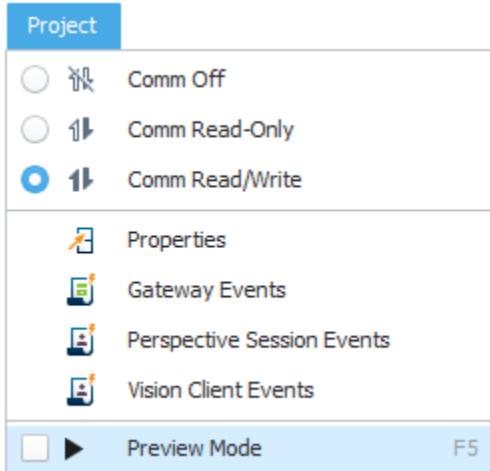
- **Console** - The [Output Console](#) is the script-writers best friend, and most frequently used to test and debug Python scripts in Ignition.
- **Image Management** - The [Image Management](#) tool manages and stores images used for your projects.
- **Script Console** - The [Script Console](#) is used to test and debug Python scripts.
- **Database Query Browser** - The [Database Query Browser](#) is a very convenient tool that lets you make simple selects and edits in a database table, and interact with all of the databases that Ignition is connected to (i.e., running queries, browsing tables and schemas). It is very common during the course of project design to inspect the database directly, or to experiment with a SQL query to get it just right.
- **Translation Manager** - Opens up the [Translation Manager](#) panel, and allows you to configure language translations. See also: [Localization and Languages](#)
- **Symbol Factory** - The [Symbol Factory](#) contains a variety of high quality vector graphics symbols that can bring your projects to life.



Previewing the Project

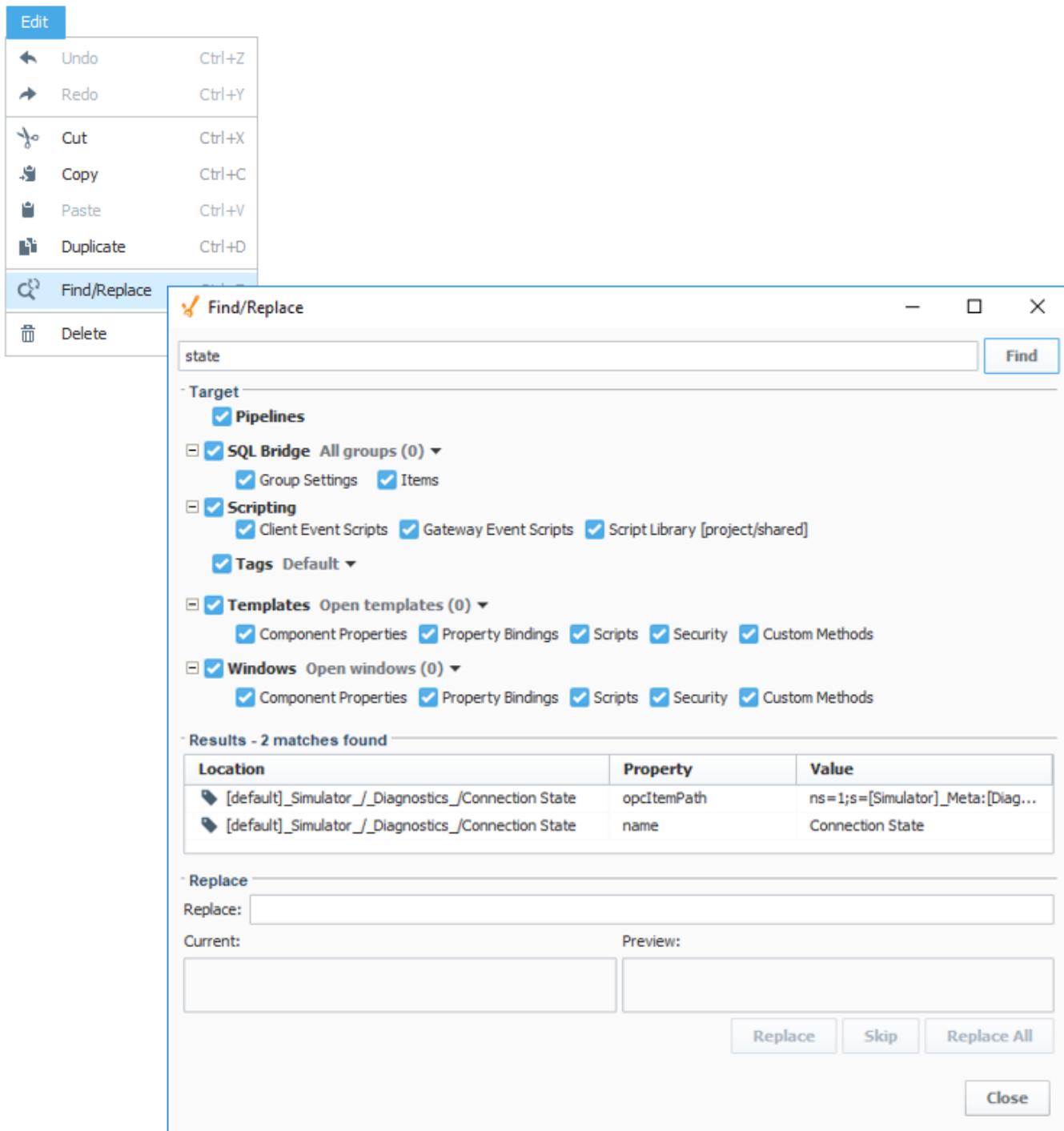
The Designer provides the capability to preview, test, and interact with the screens and functionality before you deploy your project. The Designer workspace operates in two distinct modes: Design mode and [Preview mode](#). Designers can easily switch between these modes to make sure their project is working as expected during the course of development by simply clicking **Preview Mode** ► icon from the top menubar, or clicking **Project** also from the top menubar, and selecting **Preview Mode**.

Note: The **Preview Mode** ► icon toggles to be a **Design Mode** ■ icon when it is pressed.



Find and Replace

The [Find and Replace](#) is a handy tool in the Designer workspace. You can search your entire project for specific components, properties, scripts, and more. You can even use the replace command to make mass changes expeditiously to a project with very little effort.



Keyboard Shortcuts

There are a lot of ways to speed up your development once you are familiar with how Ignition works. There are many [keyboard shortcuts](#) in Designer that are listed throughout the Designer interface alongside menu options.

	Undo	Ctrl+Z
	Redo	Ctrl+Y
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
	Duplicate	Ctrl+D
	Find/Replace	Ctrl+F
	Delete	Delete

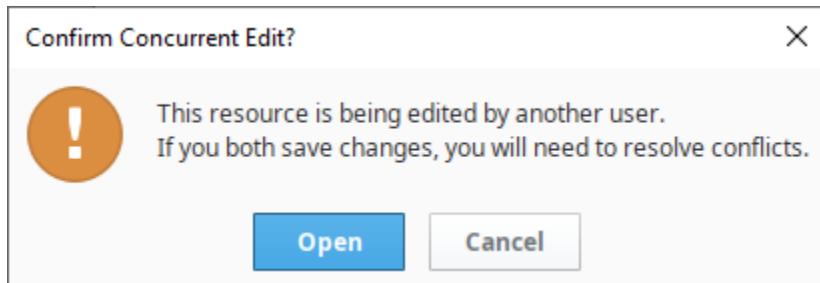
Concurrent Editing and Conflict Resolution

The Ignition Designer uses a lock-free strategy for handling concurrent editing. Multiple people can work together and make changes to a project at the same time. An unlimited number of Designers can be open concurrently, and modifying any resource in the Designer doesn't lock it. The Designer keeps track of the resources that are being edited, and any conflicting edits will be resolved at the time the project is saved. The developer who creates a saving conflict will be prompted to resolve the conflict by choosing whether to use their own changes, accept the other designer's changes, or cancel their save and figure out what to do in another way.

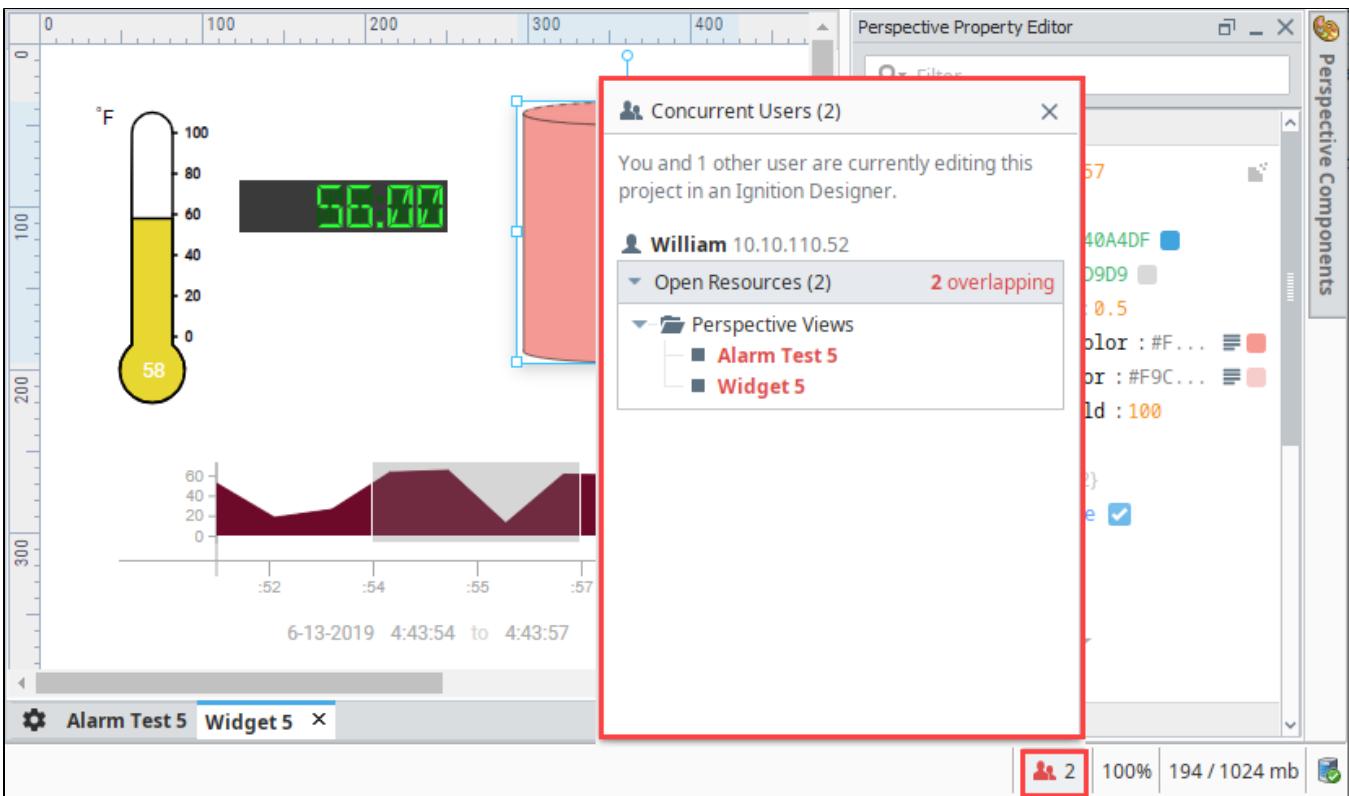
Concurrent Users UI

The Concurrent User Interface allows users to see which project resources are open in other Designer instances, the names of the users that have them open, and when a project update is available. The UI is located in the lower right corner of the Designer.

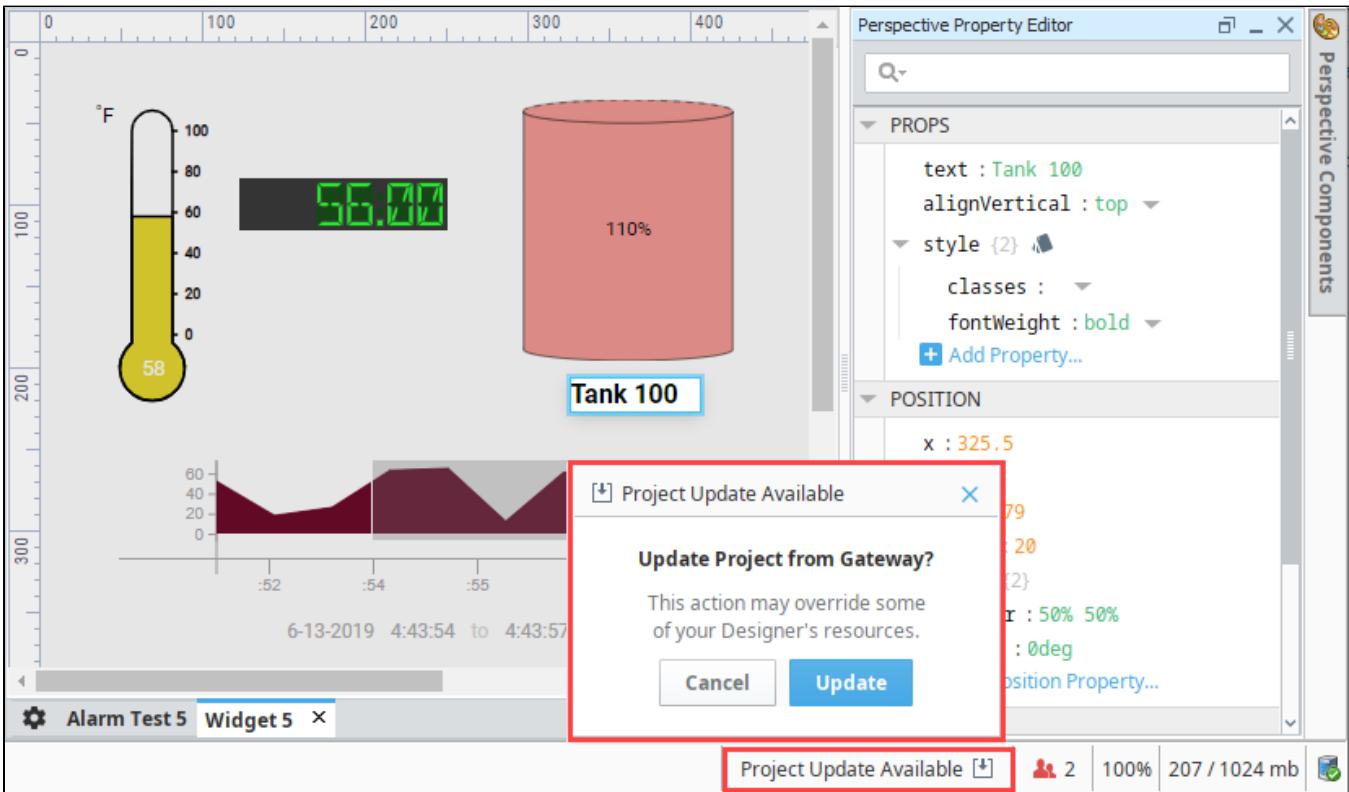
If you open a resource that is already open in another Designer, you will be greeted with a new popup confirming that you want to proceed.



The UI makes use of the color red to denote when there are conflicts with the changes in another Designer instance, or if their users are saving on overlapping resources (multiple users are making changes and saving while the same resource is open).

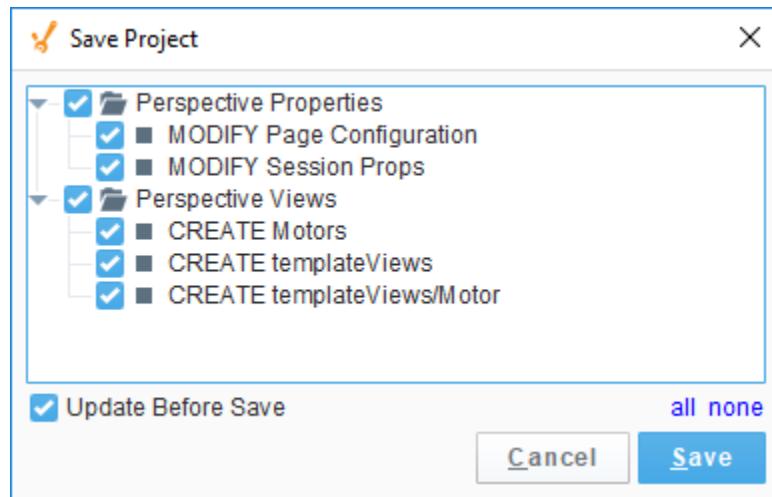


If the user in the other Designer instance saves their project, the UI will display that a project update is available. Click on **Project Update Available** to either update or cancel the project update.



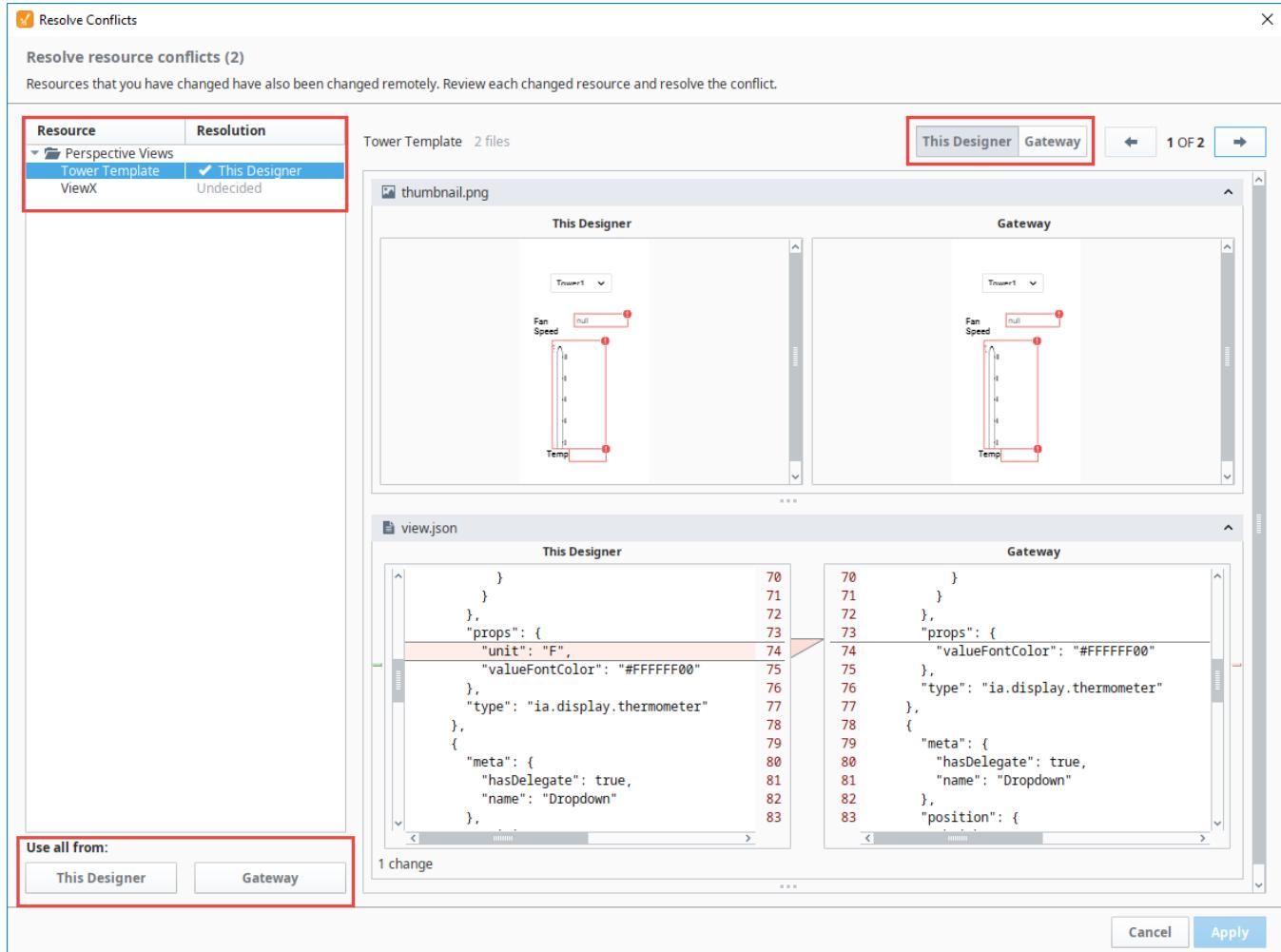
Conflict Resolution

When you're ready to save your project, go to the Menubar and select **File Save**. When the Save Project dialog pops up, select the items you want to save and click the **Save** button.



If there are any conflicts, the Designer will ask you to update your project and the conflict resolution screen will open automatically. You will see a thumbnail image of both screens and the .json code showing the conflict. To resolve the conflict, you will be given a choice; accept your changes, accept the other developer's changes, or cancel the save and contact the other developer. Buttons are provided at the top and bottom of the screen for you to enter how you want to resolve the conflict.

The Resource Tree displays the conflicts and the resolutions. The selected conflict will be highlighted in blue. The image below shows changes were made to the Thermometer. The line number and the code are highlighted at the bottom so you can quickly identify the conflict. Once the conflict is resolved, a checkmark will appear in the Resolution column next to the conflict and how it was resolved. You'll notice that there is second conflict. To resolve the conflict, select it in the Resource Tree or click the arrow to navigate to the next conflict.



Related Topics ...

- [Designer Launcher](#)
- [Perspective Designer Interface](#)
- [Vision Designer Interface](#)

In This Section ...

General Designer Interface

Designer Spaces

The Designer has a lot of panels and menus that allow you to build out a project tailored to your needs. However, while some of these like the File Menu are shared throughout the Designer, some of the panels and menu options are specific to certain objects and will typically only be displayed when an object of that type is selected. For example, when editing a Vision Window, the Designer has the Property Editor and Component Palette panels, but when editing an Alarm Pipeline, the Pipeline Blocks and Pipeline Block Editor panels are displayed instead. This creates different Designer Spaces that are used for different areas of a project.

There are many dockable and draggable panels that surround the workspace, as well as the familiar menu bars and toolbars. The dockable panels can be rearranged as you wish and will snap into place as you move them around the screen. Each workspace remembers its layout, which is the docking arrangement of the panels around it.

Docking System

The Designer's docking system provides a very flexible user interface allowing you to customize the layout as you wish. To rearrange the dockable panels, simply drag on their title bars. As you drag the panel, it will try to snap into place and show you a highlighted border. Use the highlighted border that appears to gauge where the panel will be moved to. Hold the CTRL key as you drag these panels to keep them from snapping into place. You can also drag these panels outside of the Designer or onto a second monitor.

Dockable panels can be in one of four modes:

On this page ...

- [Designer Spaces](#)
- [Docking System](#)
- [Project Browser](#)
 - [Project Resources](#)
 - [Right-Click Menu](#)
- [Tag Browser](#)
- [Menubar](#)
 - [File Menu](#)
 - [Project Menu](#)
 - [Previewing the Project](#)
- [Tools Menu](#)
- [Help Menu](#)



INDUCTIVE
UNIVERSITY

The Designer User Interface

[Watch the Video](#)

1. **Docked** - The panel is visible, and located somewhere around the perimeter of the workspace. If two panels are docked in the same location, a tab strip will appear to switch between the two panels.
2. **Floating** - A panel can be dragged outside of the workspace perimeter to be floated. The panel can now be positioned anywhere on your desktop.
3. **Pinned** - Pinning a panel makes it minimize to one of the four sides of the Designer, represented by a small tab. Hover over the tab to use the panel.
4. **Hidden** - A hidden panel is not shown. You can open it again by selecting it in the **View > Panels** menu.

Toolbars can also be rearranged and floated to your liking. Simply drag on the "textured" left edge of the toolbar. If you re-arranged your panels into a layout that you don't like, you can quickly revert back to the default by selecting the **View > Reset Panels** option from the menu bar.



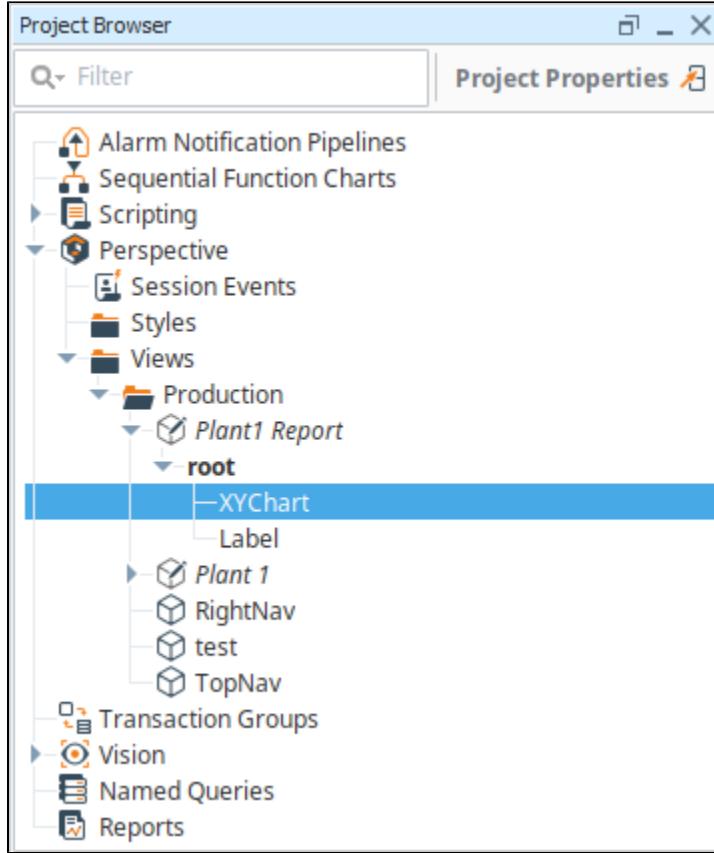
Expert Tip

Your docking preferences are stored under `%USER_HOME%/.ignition/*.layout`. If you really want to reset your preferences, remove these files and restart the Designer.

Project Browser

The **Project Browser** panel allows you to view the different Designer Spaces, and their component hierarchies at a glance. By clicking **Project Properties** icon you can view or change many of the properties settings in your project. You can expand the folders and navigate down through each folder to see elements of your project such as windows, views, containers, and components. The Project Browser shows the entire tree structure from the project level folder down to the component level. You can navigate within your project in two ways, by selecting nodes in the Project Browser tree or directly clicking on an element like a component in the project workspace. Regardless of how you select an element, properties for that element

are displayed in the Property Editor.



Project Resources

All projects have resources, such as Vision Windows, Vision Templates, Perspective Views, project scripts, reports, and named queries. Project resources can be inherited from one project to another. For more information, see [Project Inheritance](#).

Naming Conventions

Project resource names cannot be blank. They must start with a letter, a numeral, or an underscore. The following characters are reserved and cannot be used in names for project resources.

```
< (less than)
> (greater than)
:
:
" (double quote)
/ (forward slash)
\ (backslash)
| (vertical bar or pipe)
? (question mark)
* (asterisk)
```

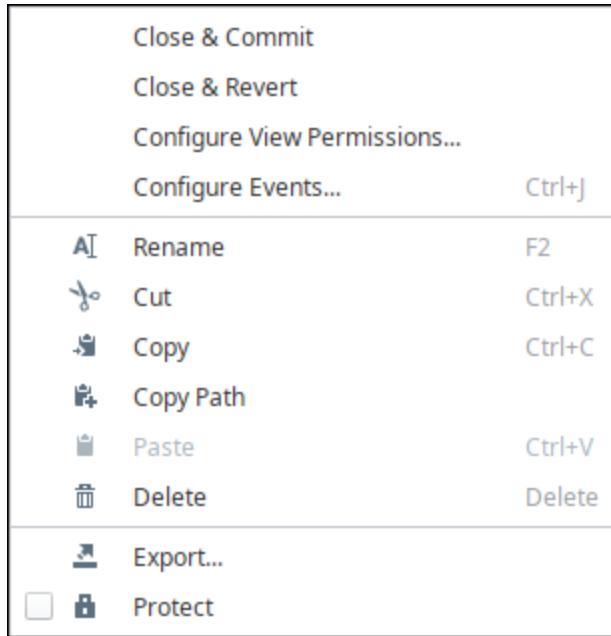
In addition, the following names cannot be used as project resource names:

CON, PRN, AUX, and NUL
COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, and COM9
LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9

Right-Click Menu

The **Project Browser** has basic functionality much the same as many applications that run on your PC. Functions can change depending on the type of project resource you are currently editing. When you right click on a project resource, several options are available:

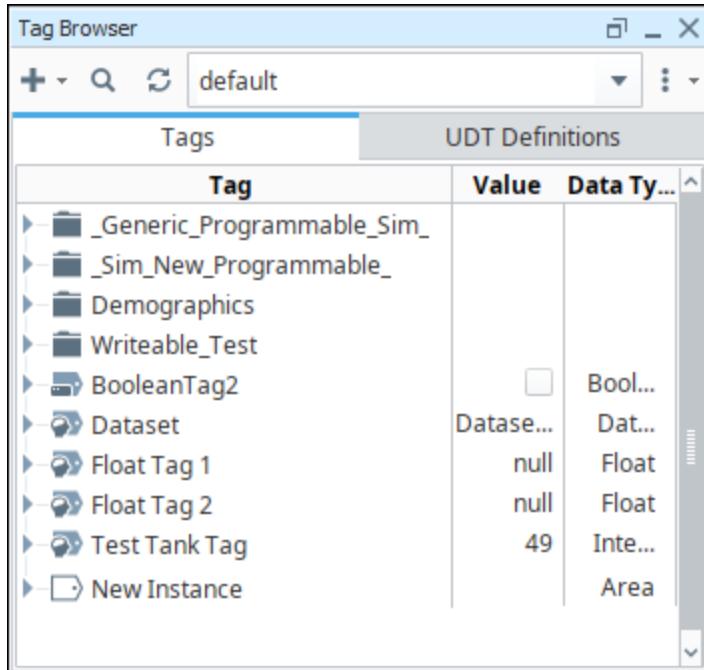
"Reverting" a modified resource will revert it back to its last saved state, assuming it's been saved before. Reverting new resources deletes them.



Function	Description
Close & Commit	Saves the updates made to the project resource and closes it.
Close & Revert	Reverts the project resource to its last saved state. Reverting an unsaved resource is the same as deleting it.
Configure View Permissions	View permissions determine whether or not a user has access to load the view.
Configure Events	Configures an event or action on a component.
Rename	Change the name of the current selection to a new name.
Cut	Cuts the current selection into the clipboard.
Copy	Copies the current selection into the clipboard.
Copy Path	Copies the path of the current selection into the clipboard.
Paste	Pastes the contents in the clipboard to the selected content.
Delete	Deletes the current selection.
Export	Exports resources from the project.
Protect	Protects the Project Resource. For more information, see Protecting Project Resources .

Tag Browser

The **Tag Browser** panel, located on the left side of the Designer workspace under the Project Browser, allows you to browse Tags in the Designer and OPC servers. Tags can be created, edited, displayed, exported, and imported directly from the Tag Browser. See [Tag Browser](#) for more detailed information.



Menubar

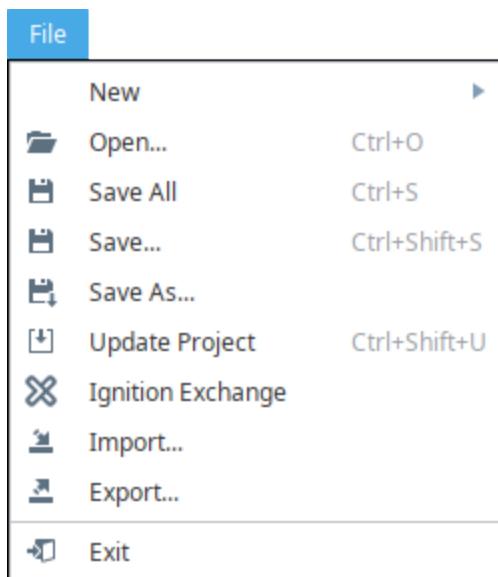
This section addresses only the functionality that the Designer menubar shares between both the Perspective and Vision modules.

For information on the menubar tabs unique to the Vision module, see [Vision Designer Interface](#).

For information on the menubar tabs unique to the Perspective module, see [Perspective Designer Interface](#).

File Menu

The **File Menu** contains most of your basic options related to saving, similar to many other software applications on your PC.

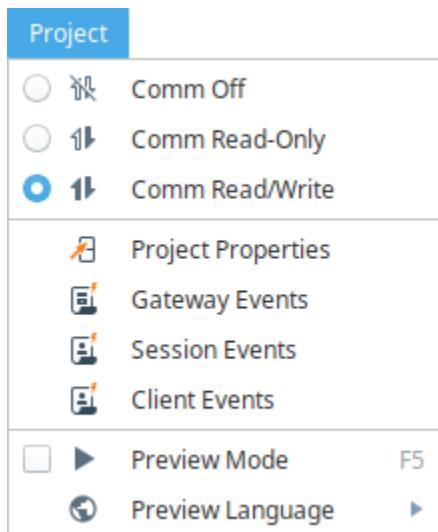


Function	Description
New	The New menu option allows you to create a new Object. That Object could be a new window, a new template, or even a new project. It can vary depending on what modules you have installed, also providing the ability to make new transaction groups,

	reports, Sequential Function Charts, and more.
Open	Allows you to Open a previously created project.
Save All	Saves all project changes, but are not pushed out to the client.
Save	Save brings up a popup menu where you can choose what to save.
Save As	Saves the current project as a new project.
Update Project	Merges any changes from the Gateway to the project, if for example there were multiple people working on the same project.
Ignition Exchange	Provides access to resources, templates, and tools that can be shared with various other industries and used in your own Ignition projects.
Import	Allows you to import or export specific resources into or out of the project such as Tags, scripts, and templates, to name a few.
Export	Exports globally scoped resources, such as Alarm Pipelines.
Exit	Exits the project and allows you to save project changes.

Project Menu

Communication between the Designer and the Gateway is controlled from the Project Menu. The Designer offers three data communication modes for your projects: **Comm Off**, **Comm Read-Only**, or **Comm Read/Write**. Comm Read-Only is the default mode which does not allow writing from the Designer to Tag or Database sources.





**INDUCTIVE
UNIVERSITY**

**About
Communication
Modes**

[Watch the Video](#)

Function	Description
Comm Off	<p>In this mode all database query traffic and Tag subscriptions and writes are blocked. Experts often use the Comm Off mode while designing a window to temporarily shut off data flow so that they can manipulate component's bound properties without the values being overwritten by the data bindings. This is useful to set the values that they want to serialize into the window. This can be important for windows with large datasets, clearing the datasets before saving the window can significantly reduce the size of the window, thus improving performance.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>Note: This setting does not affect the execution of a project's Transaction Groups because Transaction Groups execute on the Gateway, not in the Designer. See also: Comm Modes.</p> </div>
Comm Read-Only (default)	<p>Tag subscriptions and SELECT queries work, but Tag writes and UPDATE/INSERT/DELETE queries are blocked. The Designer starts up in Comm Read-Only mode as a safety mechanism so that you don't inadvertently write to a Tag as you are designing. You can customize the Designer's startup mode, see the Designer/General Properties section.</p>

Comm Read-Only (default)	All data is passed through to the Gateway. The mode can be switched at any time via the tri-state toggle selection in the main toolbar, or the radio buttons in the Project menu. A common beginner mistake is to forget to switch the mode to Comm Read/Write when attempting to test a window's functionality in Preview mode.
Properties	Opens up the Project Properties window, allowing project settings to be changed. See also: Project Properties
Event Scripts	Opens up the appropriate event script window, either client, session, or Gateway. These can also be accessed from the Project Browser. See also: Client Event Scripts and Gateway Event Scripts .
Preview Mode	Puts the Designer into Preview Mode, allowing you to interact with it like a client. See also: Previewing the Project .
Preview Language	Determines the language that the Designer will revert to when in Preview Mode. See also Localization and Languages and Localization in Vision .

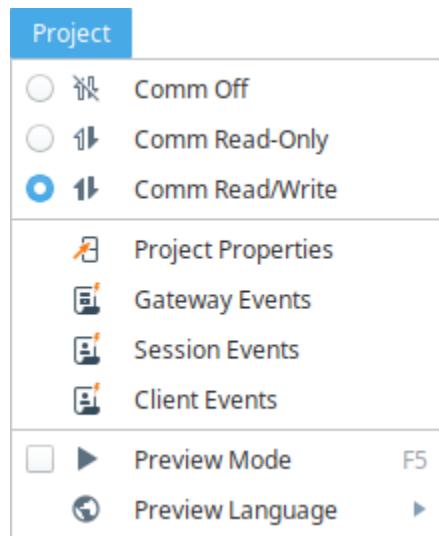
Set or Change the Communication Mode

There are three ways to set or change the comm mode in the Designer:

- Using the **Project** menu.
- The buttons located on the **Main Toolbar**.
- Using the **Project Properties** window.

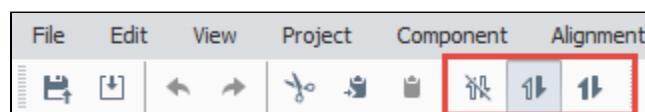
Setting the Comm Mode from the Project Menu

On the Designer toolbar, go to the **Project** menu and choose: **Comm Off**, **Comm Read-Only**, or **Comm Read/Write**.



Setting the Comm Mode from the Main Toolbar

If the Main Toolbar is enabled, go to **Project**, and you'll see a corresponding button for each Comm Mode. The currently selected Comm Mode will have its button highlighted in gray.

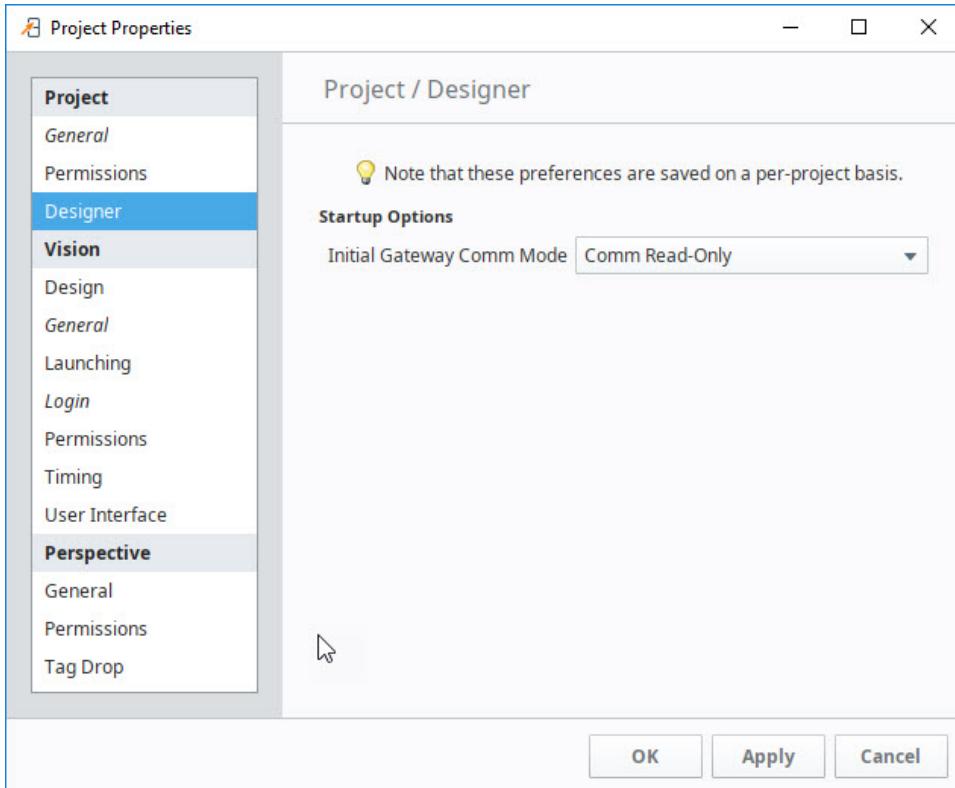


Setting the Comm Mode from the Project Properties Window

This will set the default Comm Mode that the Designer starts up in for the current project.

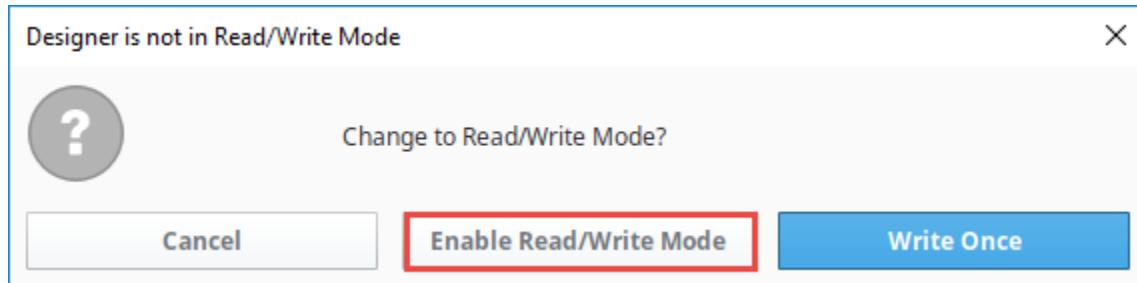
1. In the Designer, either double-click on the **Project > Properties** node in **Project Browser**, or click on the **Project > Properties** command on the top menu.
The **Project Properties** window is displayed.
2. Go to **Project > Designer**.
3. Under **Startup Options**, for the **Initial Gateway Comm Mode**, choose from the dropdown **Comm Off**, **Comm Ready-only**, or **Comm Read/Write**.

Note: These property settings are saved on a per-project basis.



Communication Error Message

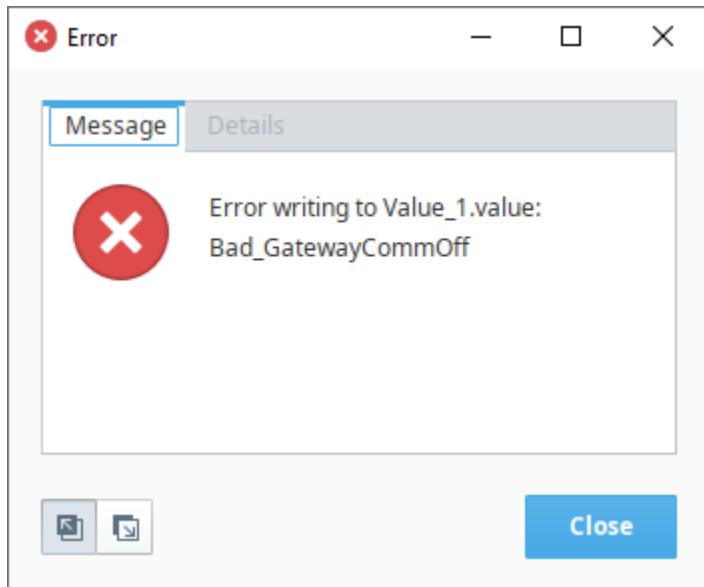
If a user is attempting to write to a project in the Designer or from a Client, and the Comm Mode is not enabled for Read/Write, a dialog box will popup stating the Designer is not in Read/Write Mode.



If your Tag is not being written too, there are a couple places to check:

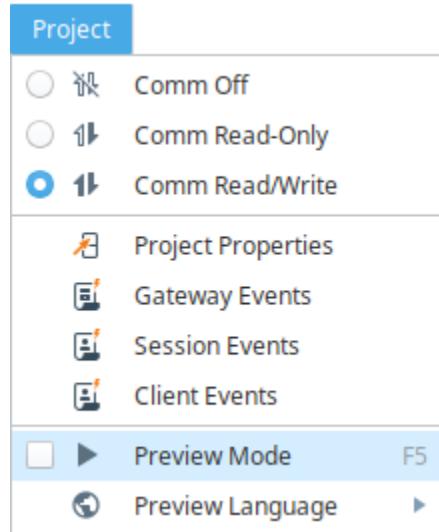
- From Main Toolbar go to **Project**, and make sure the Comm Mode is set to Read/Write.

- If Comm Read/Write is checked, you may have a Client Event Script enabled that could be preventing you from writing to a project or a Tag, as shown in the error message example below. You may need to edit your Client Event Script.



Previewing the Project

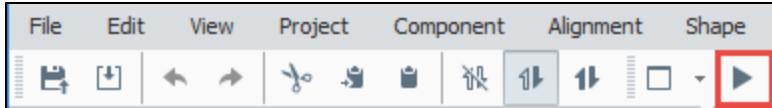
Many times, it is useful to test the components on the screen to ensure that certain bindings or scripts are working the way that was intended. The Designer can go into **Preview Mode** that will allow you to interact with the currently opened window as though you were working in a client. This means that instead of clicking between components and seeing their properties, you will be able to interact directly with the components such as clicking on a button to execute its script, or entering a value into a text field to update a Tag value.



Preview Mode

The window workspace operates in two distinct modes: Design Mode and Preview Mode. There are three different ways you can switch between Design and Preview Mode:

- One of the easiest ways is from the Main Toolbar using the **Preview**  icon or the **Design**  icon to switch between modes.



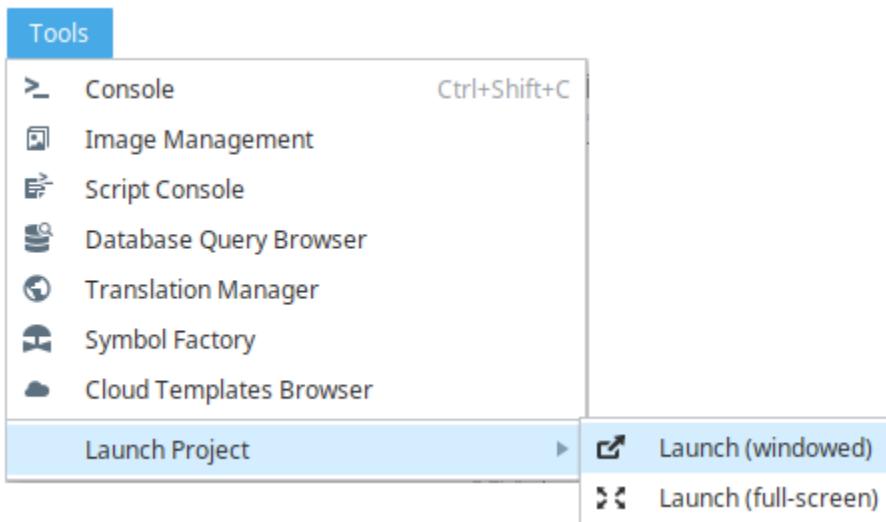
- From the Main Toolbar using the **Project > Preview Mode** menu item.
- Using the **F5** key to toggle between the two modes.

In Design Mode, your mouse is used to manipulate components in a window. You can select, drag, and resize them. You can alter data bindings and event script configuration. Data bindings are active in Design mode, but event handlers are not.

In Preview Mode, you are interacting with a "live" version of the window. Property bindings and event handlers will run, just like in the Client.

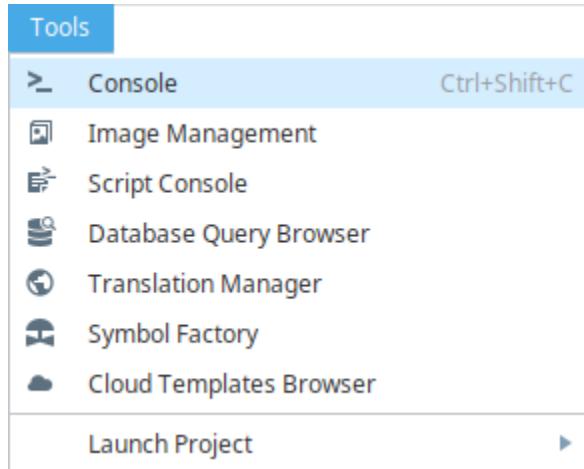
Note: Some of Ignition's functionality will not work in Preview Mode, for example, the retarget and openWindowInstance scripting functions must be tested in a Client.

Preview Mode is useful for a quick check of the operation of a window, but it becomes cumbersome when trying to test a whole project. For that, we recommend having a launched Client up as well, and doing testing in the true Client. You can quickly launch a client in one of the following two launch modes via the **Tools > Launch Project** menu.



Tools Menu

The **Tools Menu** provides some tools to help you when creating projects.

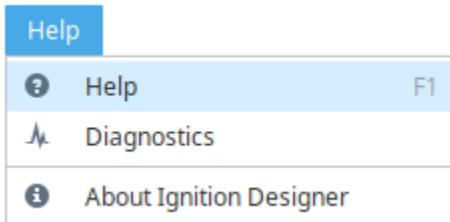


Each of the tools are described here.

Function	Description
Console	The Output Console is a dockable panel prints system messages that are coming from the designer. This can vary from simple info messages that show when things are loaded, to error messages when something goes wrong. The Console is also used frequently to test and debug Python scripts, as print statements on components that are run in the designer are printed here.
Image Management	<p>The Image Manager is available from the Tools > Image Management menu. This tool is a drag-and-drop browser that helps manage the images that are stored on the Gateway. It is important to realize that these images are shared across all projects: they are not stored inside a project itself. Use the Image Management tool to do common tasks like uploading new images and creating folders. You can drag images and folders from your computer's desktop or hard drive into this window to easily upload new images to the Gateway.</p> <p>You can also get to this tool by putting an Component Palette > Display > Image component on a window, and clicking the browse button on the image's Image Path property. See Images and SVGs in Vision and Images, SVGs, and Icons in Perspective.</p>
Script Console	Opens up the Script Console . window. This is where you can test scripts out.
Database Query Browser	Opens up the Database Query Browser panel, which allows you to run SQL queries against your database connections.
Translation Manager	Opens up the Translation Manager panel, which allows you to configure translations. See Localization and Languages .
Symbol Factory	<p>If you have the Symbol Factory module installed, you'll be able to open the Symbol Factory browser under the Tools menu in the Designer. You can browse through the symbols or use the convenient search function to find the symbol you need.</p> <p>Once you find a symbol, you can drag-and-drop it into a window. Each symbol is dropped as a shape group. You will be able to ungroup it or double-click into the group just as if you had drawn the symbol yourself using fundamental shapes. This means that you can alter the shape if you need to, or bind any colors inside the shape to a Tag to make the shape dynamic.</p>
Launch Project	Allows you to launch the project directly from the Designer. You can either launch a staging or published windowed client, or a published full screen client.

Help Menu

The **Help Menu** provides online assistance when looking for information or troubleshooting an issue.



Functions	Description
Help	Opens up your web browser and takes you to this User Manual for quick reference.
Diagnostics	The Help menu in the Designer and the Vision Client has a Diagnostics window that contains a number of tabs each providing a useful troubleshooting feature. You can right-click on any of the tabs to show or hide the other tabs. For more information on these tabs and the troubleshooting features, go to Designer Diagnostics .
About Ignition Designer	Provides information about the Designer such as the versions of the modules, Java version, and Gateway IP Address that the Designer is using.

Related Topics ...

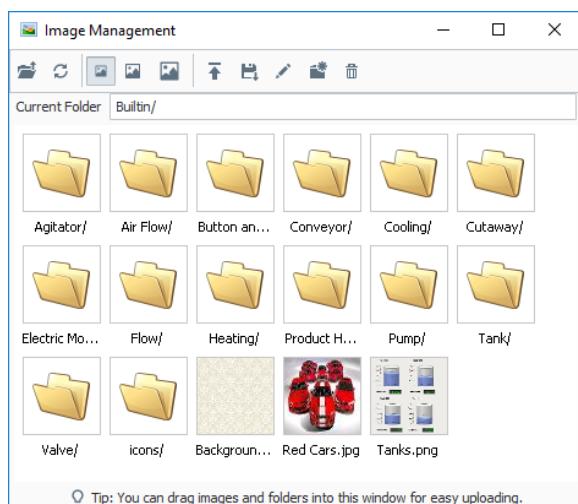
- [Vision Designer Interface](#)
- [Perspective Designer Interface](#)

Designer Tools

The Designer comes with many tools that allow you to manage and test various resources within a project. Each of the tools have their own interface and are accessed within the Tools menu on the menu bar of the Designer.

Image Management Tool

The [Image Management Tool](#) allows you to manage the images that are stored within the Ignition Gateway. The path to the images can be copied out and pasted into a component's property that is expecting an image path.

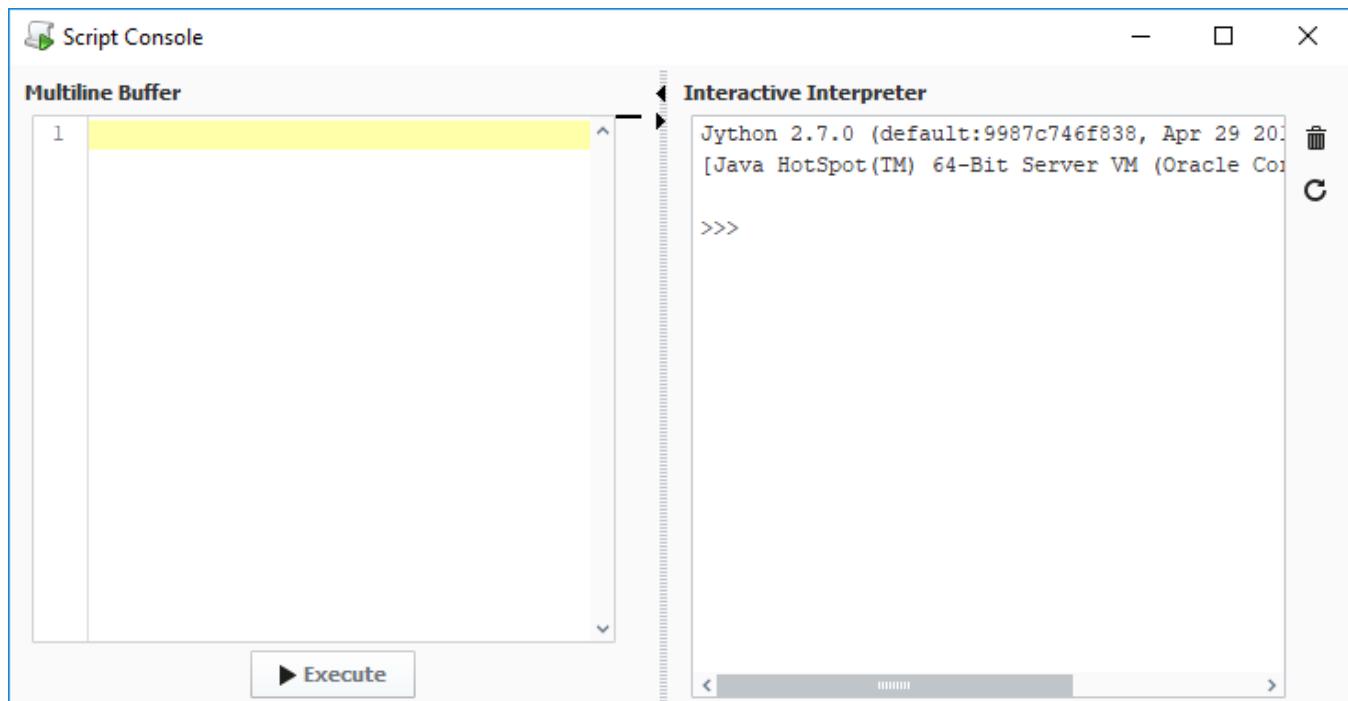


On this page ...

- [Image Management Tool](#)
- [Script Console](#)
- [Database Query Browser](#)
- [Translation Manager](#)
- [Symbol Factory](#)

Script Console

The [Script Console](#) allows you to test out code snippets, printing the results out in the panel on the right.



Database Query Browser

The [Database Query Browser](#) is like the Script Console, but for SQL and databases. Here, you can test out queries to ensure you are returning the correct data from your database.

The screenshot shows the Database Query Browser interface. At the top, there is a query editor window containing the SQL command: `SELECT * FROM audit_events`. To the right of the query editor is an "Execute" button with a yellow lightning bolt icon. Below the query editor, there is a checkbox labeled "Limit SELECT to: 1000 rows".

Below the query editor is a results pane titled "Resultset 1" and "Resultset 2". The "Resultset 1" pane displays a table with columns: AUDIT_EVENTS_ID, EVENT_TIMESTAMP, ACTOR, ACTOR_HOST, and ACTION. The table contains 23 rows of data. The "Resultset 2" pane shows the schema for the audit_events table, listing columns: ACTION (VARCHAR), ACTION_TARGET (TEXT), ACTION_VALUE (TEXT), ACTOR (VARCHAR), ACTOR_HOST (VARCHAR), AUDIT_EVENTS_ID (INT), EVENT_TIMESTAMP (DATETIME), ORIGINATING_CONTEXT (TEXT), ORIGINATING_SYSTEM (TEXT), and STATUS_CODE (INT).

At the bottom of the results pane, it says "23 rows fetched in 0.004s" and has buttons for Auto Refresh, Edit, Apply, and Discard.

Translation Manager

The [Translation Manager](#) allows you to add, edit, and remove translation mappings to your system. Works in conjunction with the [Localization](#) system.

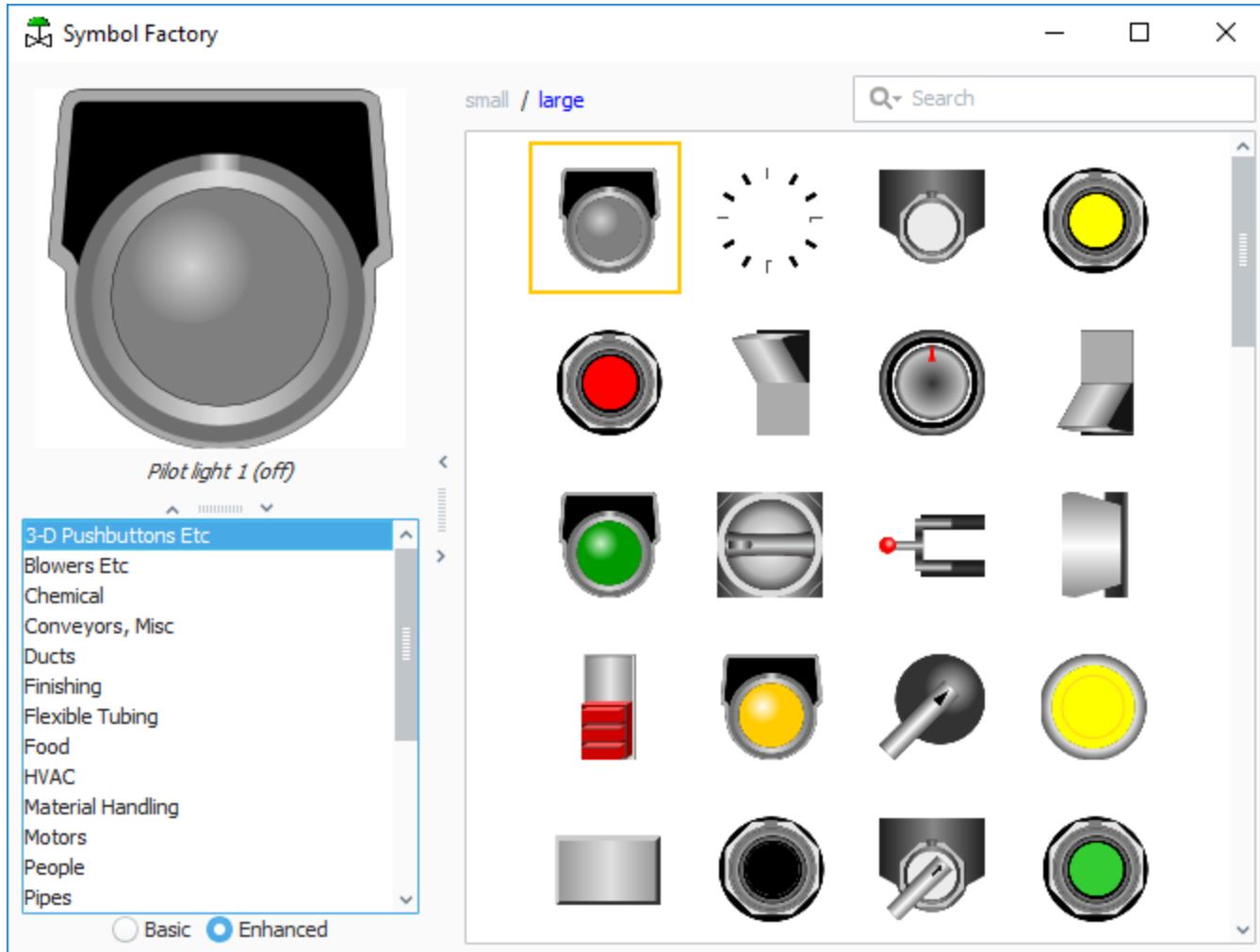
The screenshot shows the Translation Manager interface. On the left, there is a sidebar titled "Languages" with checkboxes for "(All)", "English", and "Spanish", all of which are checked. Next to the sidebar is a "Translation Terms" section with a search bar and a table. The table has three columns: "Key", "English (Alternate)", and "Spanish". The table contains the following data:

Key	English (Alternate)	Spanish
button		buton
component		componente
industry		industria
level		nivel
tank		tanque
temperature		temperatura
user		usuario

To the right of the table are several icons for managing translations: a plus sign for adding, a minus sign for removing, a magnifying glass for search, a file icon, and a wrench icon.

Symbol Factory

[Symbol Factory](#) module is a unique designer tool included with Vision or Perspective.



Related Topics ...

- [Images and SVGs in Vision](#)
- [Localization and Languages](#)

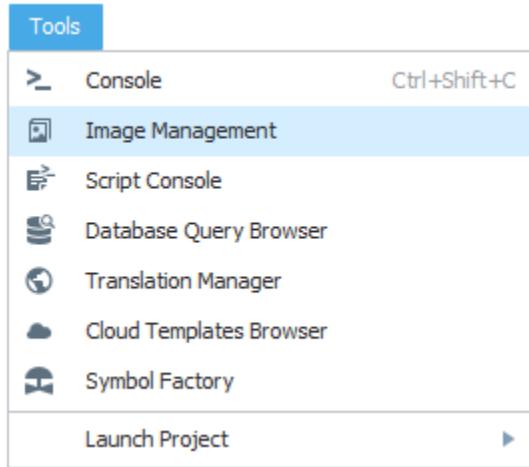
In This Section ...

Image Management Tool

Images such as PNGs, JPGs, GIFs, and SVGs can be uploaded to the Image Management Tool and used inside of windows in Ignition. Once uploaded, these images may be used on windows and in templates. The Image Manager tool, available from the Tools > Image Management, provides an interface to upload, download, or select images.

Note: The Image Management tool does not support bitmap files.

SVGs are supported in the Image Management Tool.



On this page ...

- [Uploading an Image to the Image Management Tool](#)
- [Downloading Images from the Image Management Tool](#)
- [Exporting and Importing Images in Projects](#)



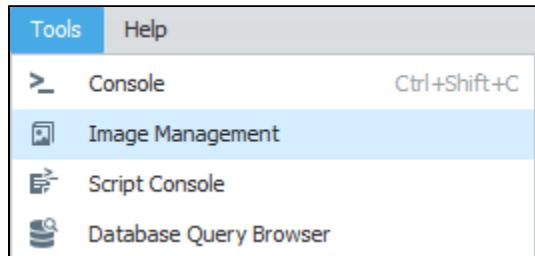
INDUCTIVE
UNIVERSITY

**Images (png, jpg,
gif)**

[Watch the Video](#)

Uploading an Image to the Image Management Tool

There are two ways to upload an image into the Image Management tool. Both ways involve having the Image Management tool open. At the top of the Designer in the Menu Bar, select **Tools > Image Management**.

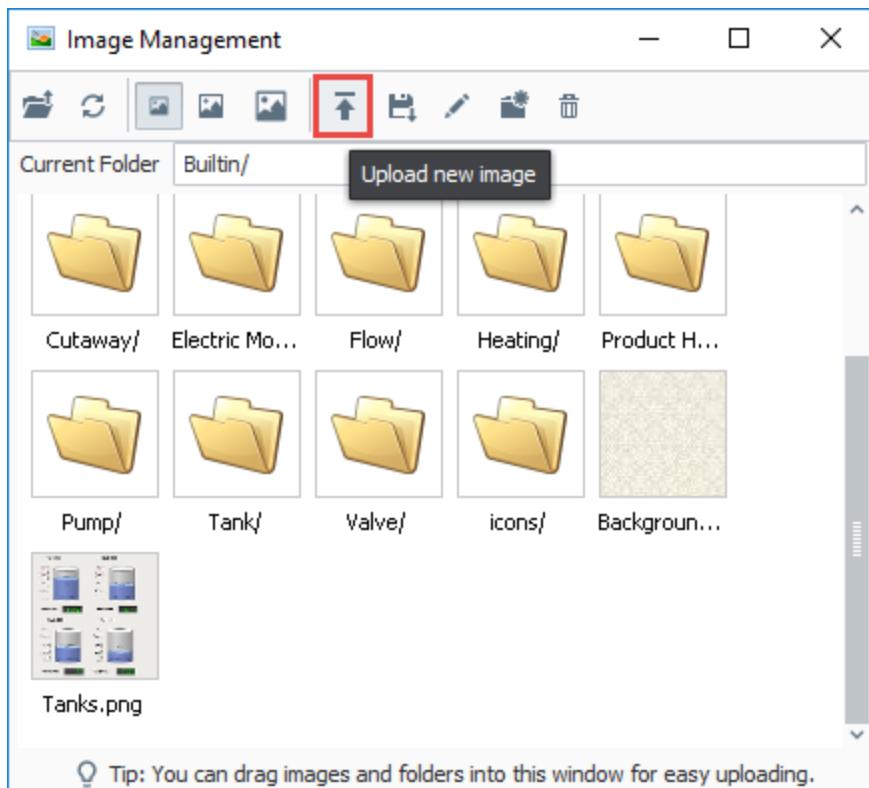


Upload on Drag and Drop

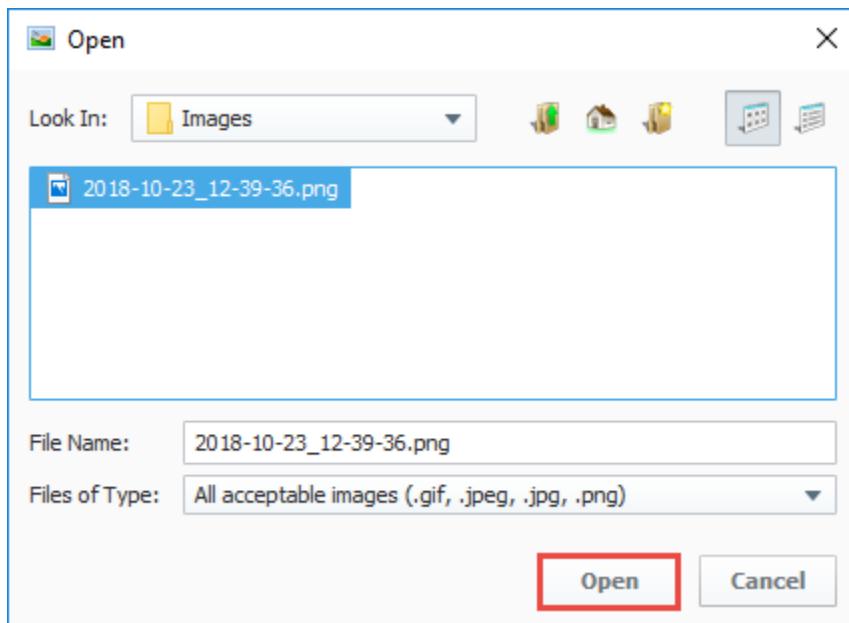
Images can simply be dragged and dropped from the local file system into the **Image Management** window.

Manual Upload

Alternatively, the Image Management window has an Upload button to pass images in. Locate the directory you wish to upload the image. You may use the root folder, or create a new folder to keep your images organized. When ready, click on the **Upload Image** button.



An **Open** dialog window will appear. Simply find your image on the local system, and click **Open** to upload the selected image.



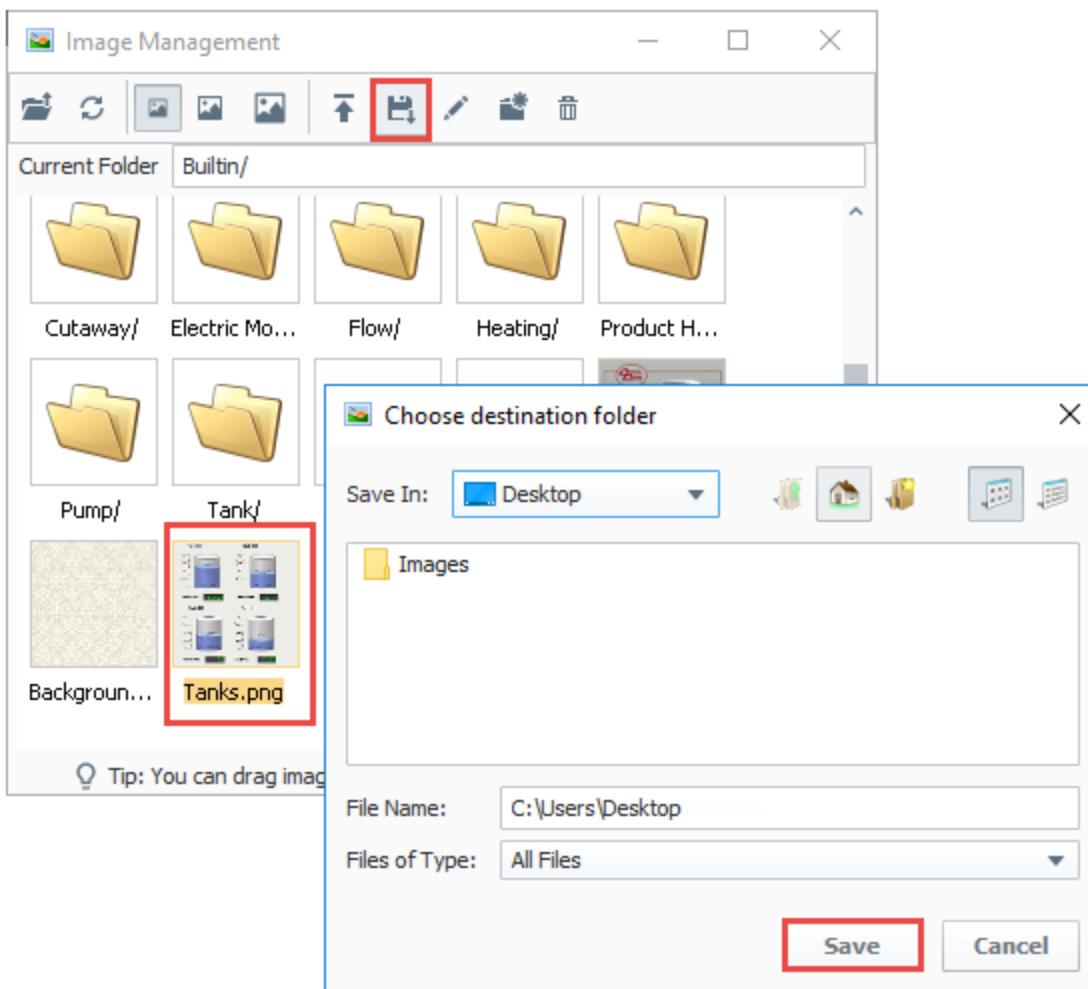
Downloading Images from the Image Management Tool

Single images, as well as entire directories, may be downloaded from the Image Management tool. This is useful when migrating a project to another Gateway.

Image downloads can be taken from either the **Image Management** or **Image Browser** windows. If at least one folder or image is selected, the **Save**



icon will become enabled. Click the **Save** icon, pick a local directory to save the images to, and click the **Save** icon again. All images and subfolders in the selected folder will be copied to the selected directory.



Exporting and Importing Images in Projects

Uploading an image involves storing the file in the Gateway's internal database. This means that project exports do not contain any referenced images.

When exporting a project for use in another Gateway, it is recommended to also export any images that the project uses, and upload them into the new Gateway at the same directory. Examples of uploading and downloading images can be found on this page.

Related Topics ...

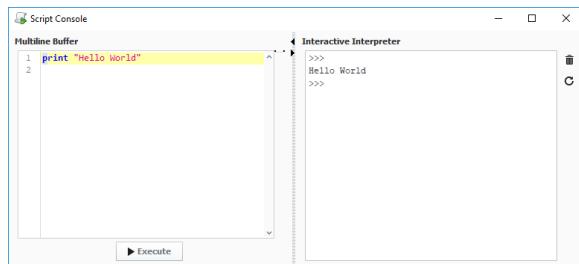
- [Images, SVGs, and Icons in Perspective](#)
- [Images and SVGs in Vision](#)
- [Using Images](#)
- [Symbol Factory](#)

Script Console

The Script Console is a live Python terminal that is only accessible in the Designer. It is a great way to very quickly test a script as it does not rest on a scripting event or specific component. The Script Console can be opened via the **Tools > Script Console** menu. It consists of two parts: a **Multiline Buffer**, and an **Interactive Interpreter**. Code can be typed into both sides.

Due to how "scope" works, the Script Console can not interact with components on a window, but it can call Project and Shared scripts. If a Project or Shared script was recently added, then the console will need to be reset before it can be called. This can be accomplished by clicking on the **Reset**  icon in the upper right.

Note: Gateway-scoped information will not appear in either the Script Console or [Output Console](#). Instead, the output will be sent to the wrapper.log file. Alternatively, `system.util.getLogger` will send messages to the Gateway Console, and is a preferred method of troubleshooting Gateway scoped scripts.



On this page ...

- Features
 - Multiline Buffer
 - Interactive Interpreter

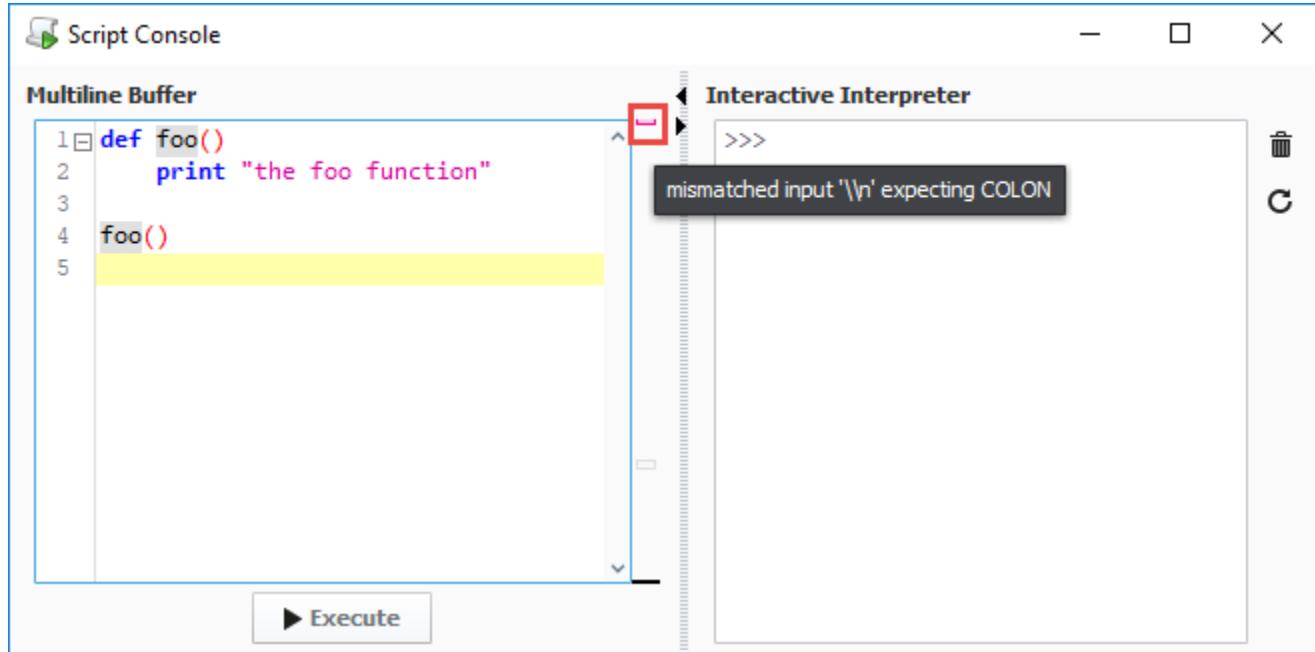


Scripting Console

[Watch the Video](#)

Features

There are several icons and user interface elements located on the Script Console window. The image below highlights a syntax error.



A reference of the icons and descriptions are found in the table below:

Icon	Name	Description
	Multiline Buffer	Code editor for writing and executing multiple lines of Python code.

	Clear	Clears the text from the Interactive Interpreter.
	Reset	Clears and resets the text, and deletes all user defined objects (variables and functions) from the Interactive Interpreter.
	Expand/Collapse	Expands / Collapses Multiline Buffer and Interactive Interpreter.
	Syntax Error Highlight	Shows up next to a line in the Multiline Buffer identifying an error. Hover over the Error Symbol to see information on the exception.

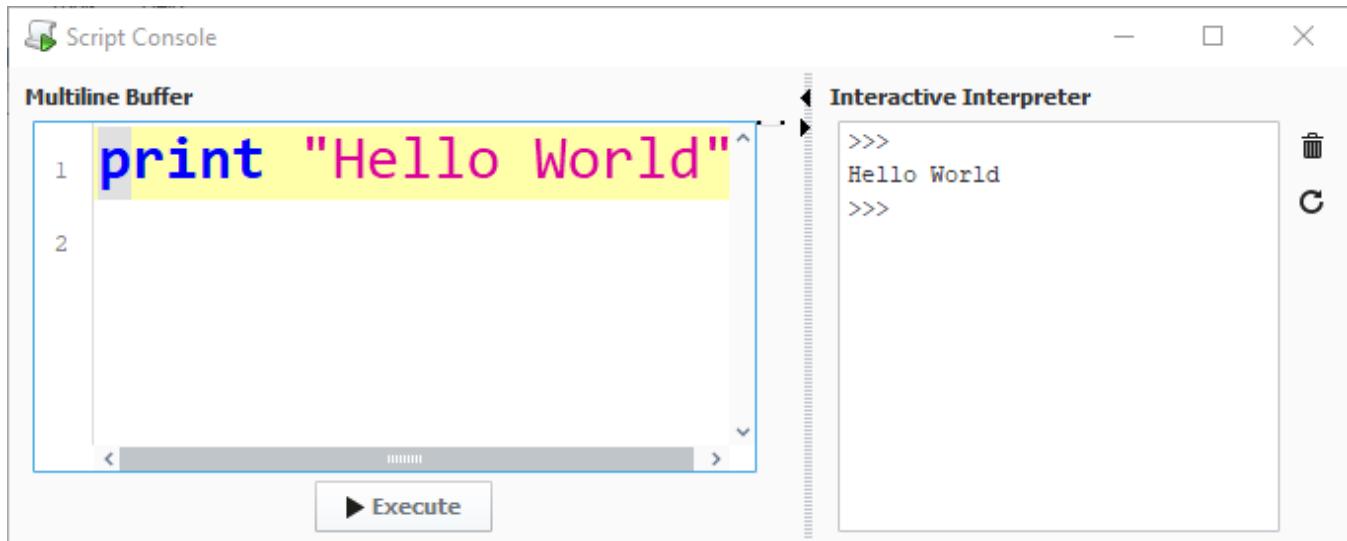
Multiline Buffer

The Multiline Buffer, located on the left side of the Script Console, allows for multiple lines of code to be entered and then executed by clicking on the button. All `print` statements will output to the Interactive Interpreter. It also supports code folding for function definitions and comments.

When executing a script in the console, the button will change to an button. Developers can press the `Interrupt` button to interrupt / stop a script from executing when testing code with a lot of data, or when the script inadvertently gets stuck in an infinite loop.

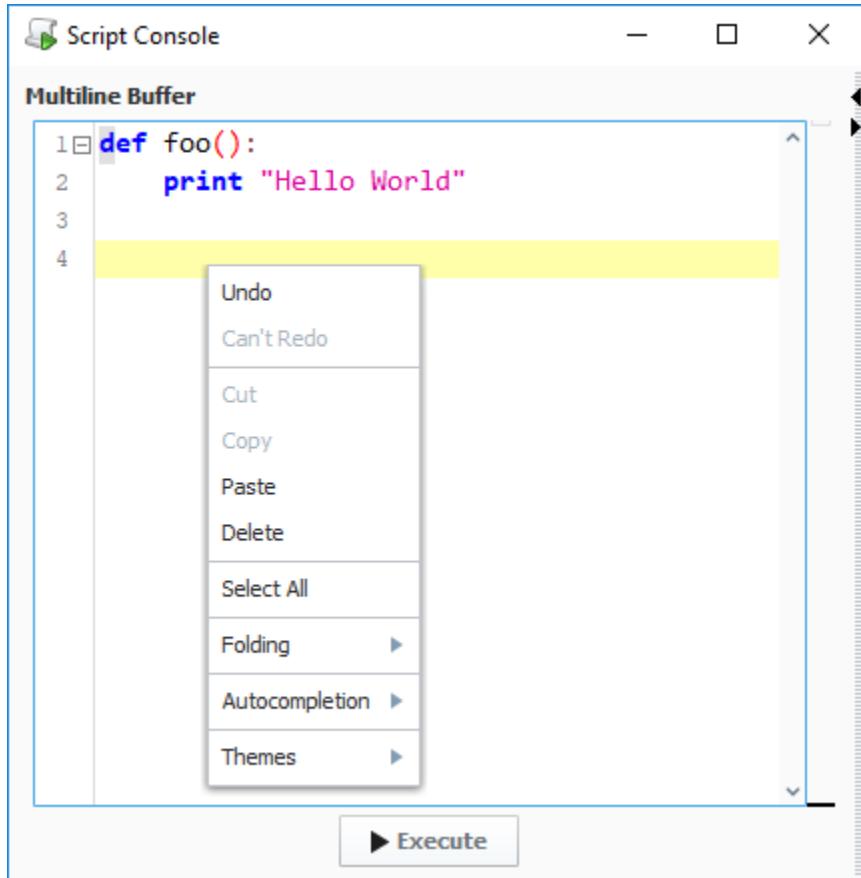
Font Size Adjustment

Font size in the Multiline Buffer can be adjusted by holding `Ctrl` and scrolling the mouse wheel.



Right-Click Menu

Right clicking on the Multiline Buffer opens a menu. The menu options are described in the table below.



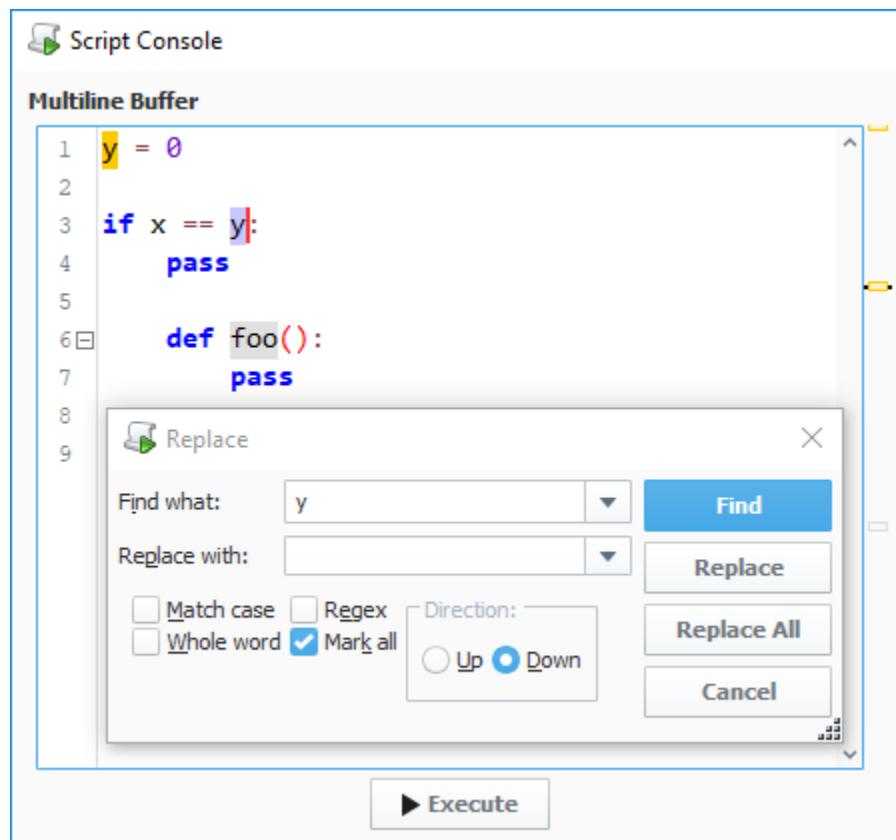
Main Menu Options	
Name	Description
Undo	Undoes the last action.
Redo	Gets rid of the last undo action.
Cut	Cuts the selected text.
Copy	Copies the selected text.
Paste	Pastes the selected text.
Delete	Deletes the selected text.
Select All	Selects all text in the window.

Folding	
Name	Description
Toggle Current Fold	Expand or collapse the fold where the text cursor is located.
Collapse All Comments	Collapse all instances of multi-line comments. Only contiguous comments are collapsible.
Collapse All Folds	Collapse all expanded folds.
Expand	Expand all collapsed folds.

All Folds	
Autocompletion	
Name	Description
Automatic Activation	Determines if the Autocompletion window to automatically appear. When set to true, the window will appear after a second of inactivity once at least "system." has been typed. When set to false, the window can still be brought up manually by pressing Ctrl+Space .
Show Description Pane	Determines if the Description pane should appear in the Autocompletion window.
Themes	
Name	Description
Default	Uses the default theme with light background, dark characters.
Dark	Uses a dark theme with dark background, light characters.
Visual Studio	Uses a theme reminiscent of Visual Studio. Light background, dark characters.

Find/Replace

Pressing **Ctrl+R** while the text cursor is in the Multiline Buffer opens a Find and Replace window. This will search for instances of text throughout the Multiline Buffer, and allows the user to replace all or some instances with new text.



Keyboard Shortcuts

The following shortcuts apply only to the Multiline Buffer.

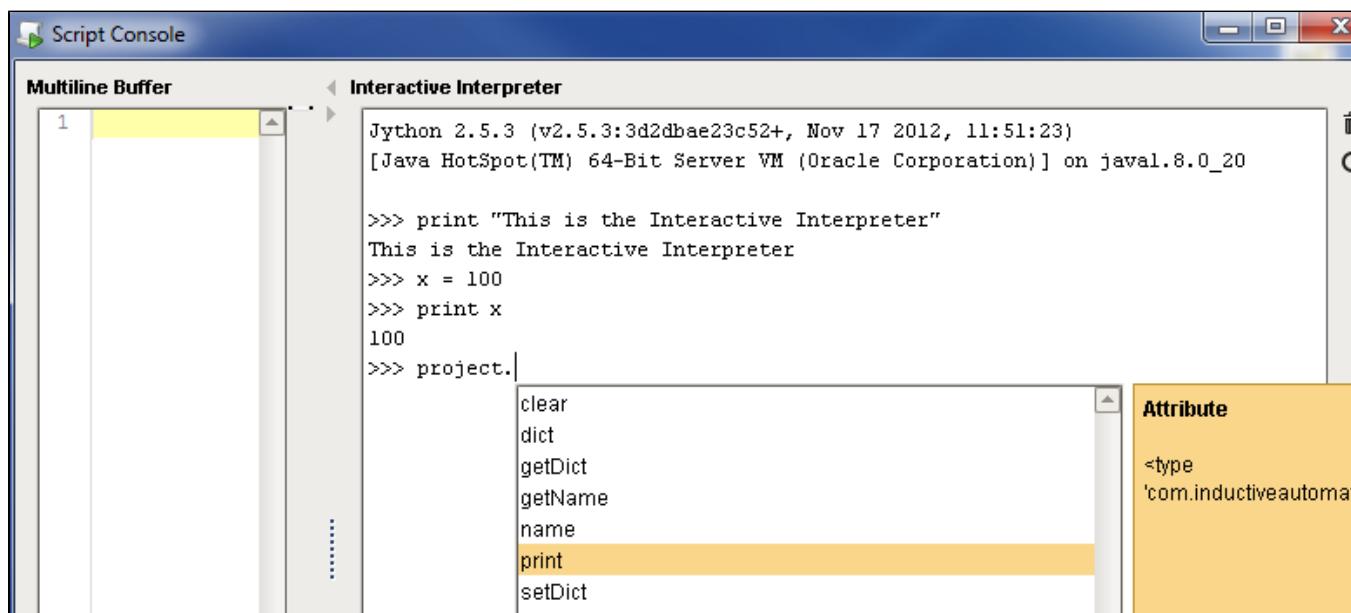
Key(s)	Description
Ctrl + </> (on the number pad)	Collapse all folds.
Ctrl + <*> (on the number pad)	Expand all folds.

Ctrl + <> (on the number pad)	Collapse the fold on the same line as the text cursor.
Ctrl + <> (on the number pad)	Expand the fold on the same line as the text cursor.
Ctrl + <Space>	Open Autocompletion window. By default, the window will automatically open once "system." has been typed.
Ctrl + <R>	Open Find/Replace window.
Ctrl + <Mouse Wheel Scroll>	Increase and decrease the font size.
Ctrl + <Enter>	Executes the script in the script editor.

Interactive Interpreter

The Interactive Interpreter is located on the right side of the Script Console, and allows you to run a single line of code at a time. Code is executed from the Interactive Interpreter by pressing the **Enter** key. Print statements from both sides of the Script Console will appear in the Interactive Interpreter.

The Autocompletion window, available in the Interactive Interpreter, has access to the current working environment so items such as Project and Shared scripts will automatically appear. They can also be typed in manually.



Keyboard Shortcuts

The following shortcuts apply only to the Interactive Interpreter

Key(s)	Description
Ctrl + <L>	Clear the Interactive Interpreter. Functionally the same as clicking the Clear button.
Ctrl + <R>	Reset the Interactive Interpreter. Functionally the same as clicking the Reset button.
Up arrow	Cycle backward through command history.
Down arrow	Cycle forward through command history.
Ctrl + <C>	Keyboard interrupt.
Ctrl + <Space>	Open Autocompletion window. By default, the window will automatically open once an "object." has been typed such as "system" or "project," and a Project script has already been defined.
Ctrl + <A>	Move the text cursor to the start of the line. Similar to pressing the Home key.

Related Topics ...

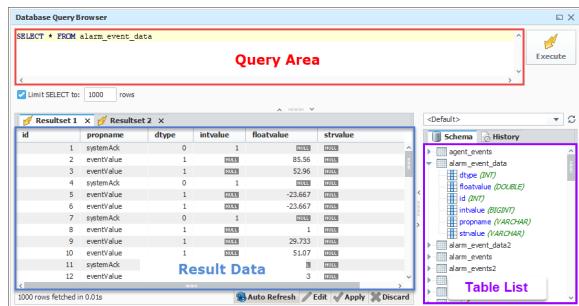
- [Output Console](#)
- [Python Scripting](#)
- [Scripting in Ignition](#)

Database Query Browser

The Database Query Browser Panel

The **Database Query Browser** is a very convenient tool that lets you query any database connected to Ignition, and interact with tables. Because Ignition is so heavily integrated with databases, it is very common in the course of project design the need to inspect the database directly, or to experiment with a SQL query to get it just right.

The Database Query Browser is found in the tools menu and has a three basic parts to it. The **Query Area** is where a query can be written and then executed against the selected database connection. This can accept any type of query, but if the query would update the database, the Designer needs to have its **Gateway Communication Mode** set to read/write first. The database connection that the query is executed against can be chosen from the dropdown, below the **Execute** button. The **Result Data** space is where the results of the executed query will appear. If a SELECT query was run, then the table data will be shown. If an UPDATE, INSERT, or DELETE query was run, then the number of rows affected will be displayed instead. Finally, the **Table List** will display all of the tables in the specified database connection. The tables can be expanded to show the columns and their data types for that table. This can help when writing queries in the Query Area. Additionally, when a table is double clicked, the Query Area will automatically be populated with a SELECT * FROM the table.



On this page ...

- [The Database Query Browser Panel](#)
- [Features of the Query Browser](#)
 - [Multiple Resultsets](#)
 - [Query History](#)
 - [Select Limit](#)
 - [Auto Refresh](#)
 - [Editing the Table in the GUI](#)



Using the Query Browser

[Watch the Video](#)

Features of the Query Browser

The Database Query Browser has a few features that can help manage and build any SQL query.

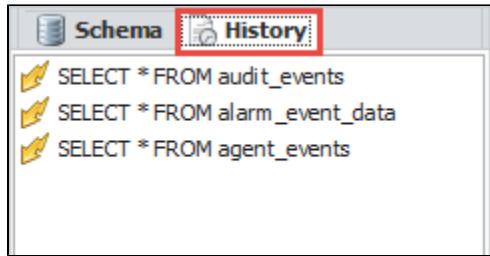
Multiple Resultsets

The Query Browser allows you to make multiple tabs of results, so that multiple queries can be run and their results compared to each other. Simply right click on the **Resultset** area and select **New Tab**. Your queries will be saved for each tab.

A screenshot of the Database Query Browser showing two result sets, 'Resultset 1' and 'Resultset 2', side-by-side. Below them is a 'New Tab' button. The 'Resultset 1' tab displays a table with columns '_EVENTS_ID', 'EVENT_TIMESTAMP', 'ACTOR_HOST', and 'ACTION'. The table has five rows. The 'Resultset 2' tab is currently empty. At the bottom of the window is a toolbar with various icons.

Query History

At the top of the Table List, there is a second tab labeled **History**. This will switch the Table List to instead show the history of queries run in the query browser. Double clicking one of the entries will push the query into the Query Area (for the selected tab). This allows you see what queries have been run previously to help you tweak your current query.



Select Limit

By default, any SELECT statement is limited to 1000 rows. This is to help the queries return quickly, however, it may not always be wanted. This can be turned off or adjusted by either clicking the checkbox or manipulating the row number located underneath the Query Area on the left. It is recommended to leave this on unless you know the result set size. It's better to use a count query than to return all results to see the result count. IE: SELECT COUNT(*) FROM table...



Auto Refresh

The Query Browser also lets you monitor a database table for changes by using the **Auto Refresh** button. This is often convenient when designing Transaction Groups. As the group runs, you can view the table that it is targeting with Auto Refresh turned ON to watch how the group is altering the table. Simply click the Auto Refresh button at the bottom of the Query Browser to periodically rerun the query in the Query Area. Make sure to include an ORDER BY clause so your results show up in the order you want.



Editing the Table in the GUI

In addition to editing the table data using INSERT, UPDATE, and DELETE statements, the data can also be edited within the Result Data. Simply click the Edit button at the bottom of the Query Browser window.



Editing in this way requires that the data be obtained from a single table with a simple query. The table must also not be tied to other tables.

Once the Edit button is clicked, the values in the table can be edited by double clicking on the value and entering a new one.

Database Query Browser

```
SELECT * FROM machines
```

Limit SELECT to: 1000 rows

Resultset 1

id	machine_name	area_number
1	Conveyor	1
2	Press	2
3	Tank	1
4	Packer	3
5	Loader	3
6	Oven 1	3
7	Oven 2	2
9	Mixer	3
10	Cold Storage	2
12	Chopper	1

10 rows fetched in 0.006s

Auto Refresh Edit Apply Discard

<Default>

Schema History

- + alarm_events
- + containers
- + files
- + machines
 - id (INT)**
 - area_number (INT)**
 - machine_name (VARCHAR)**
- + scada_roles
- + scada_user_ci
- + scada_user_ex
- + scada_user_rl
- + scada_user_sa
- + scada_users

last data 11-2017 11

Right clicking on a row also provides a few options:

- **Add Row:** Will add a new row to the table for data to be entered into.
- **Clear Field:** Will clear out the value in the selected cell so that it will be NULL. This is different than simply deleting the value out and leaving it empty.
- **Delete Row(s):** Will delete the selected row or rows from the table.
- **Copy Row Values:** Will copy the row values in a comma separated form to the clipboard.
- **New Tab:** Will Create a new Resultset tab for a new query to be run in.

When editing values, cells will highlight depending on what is being done to them. Green cells are new, and typically indicate a new row was added. Red cells are marked for deletion, and will be deleted when the changes are confirmed. Blue cells are cells that have had values changed during editing.

Database Query Browser

```
SELECT * FROM machines
```

Limit SELECT to: 1000 rows

Resultset 1

id	machine_name	area_number
1	Conveyor	1
2	Press	2
3	Tank	1
4	Packer	3
5	Loader	3
6	Oven 1	3
7	Oven 2	2
9	New Mixer	3
10	Cold Storage	2
12	Chopper	1
0		0

10 rows fetched in 0.006s

Add Row

- Clear Field**
- Delete Row(s)**
- Copy Row Values**
- New Tab**

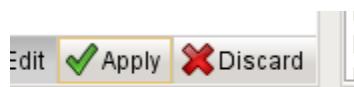
Discard

<Default>

Schema

- alarm_events
- containers
- files
- machines**
 - id (INT)**
 - area_number (INT)
 - machine_name (VARCHAR)
- scada_roles
- scada_user_ci
- scada_user_ex
- scada_user_rl
- scada_user_sa
- scada_users
- scd_data_11_2017_11

After making edits to the table data, the changes then either need to be applied or discarded. This gives you the opportunity to revert the table to the way it was before the current edit session, or apply the changes and rewrite the table appropriately. Simply click the corresponding button at the bottom of the Query Browser next to the Edit button. Make sure your Gateway Communication Mode is set to Read/Write before Applying your changes.

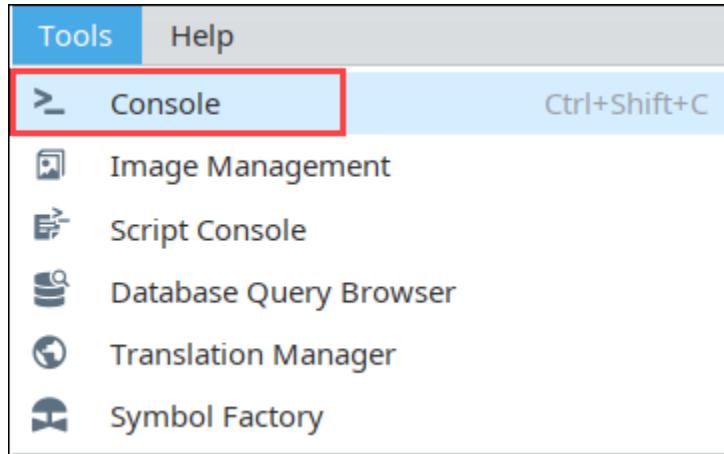


Related Topics ...

- [Keyboard Shortcuts](#)

Output Console

The Output Console is a dockable panel that you can open via Tools > Script Console menu or the Ctrl-Shift-C keyboard shortcut. The Output Console prints system messages that are coming from the Designer. This can vary from simple info messages that show when things are loaded, to error messages when something goes wrong. The Console is also frequently used to test and debug Python scripts, as print statements on components that are run in the Designer are printed here.



For example, by using the `print` function in your script, you can observe the inner workings of your script as it executes. If you executed the following script:

```
# A Python script that demonstrates the print statement.

print "Print me first"
x=10
z=2
print x, y, x/y

#Result looks like this:
#print me first
#10 2 5
```

Note: Gateway-scoped information will not appear in either the Output Console or [Script Console](#). Instead, the output will be sent to the [wrapper.log file](#). Alternatively, `system.util.getLogger()` will send messages to the Gateway Console, and is a preferred method of troubleshooting Gateway scoped scripts.

The Output Console is most frequently used to test and debug Python scripts on components in Ignition. By using the `print` keyword in your script, you can observe the inner workings of your script as it executes. For example, if you executed the following script:

Python - Using the Output Console to Test and Debug Python Scripts

```
# A function that intercepts tag writes, printing out the previous value first.
def writeToTag(path, value):
    prevValue = system.tag.getTagValue(path)
    print "Writing value '%s' to %s, was previously '%s'" % (value, path, prevValue)
    system.tag.writeToTag(path, value)

writeToTag("Compressor/HOA", 2)
```

It would print the following to the console:

Writing value '2' to Compressor/HOA, was previously '0'

Note: The Output Console is also available in the Vision Client from **Help > Diagnostics** and selecting the **Console** tab.

Related Topics ...

- [Python Scripting](#)
- [Scripting in Ignition](#)

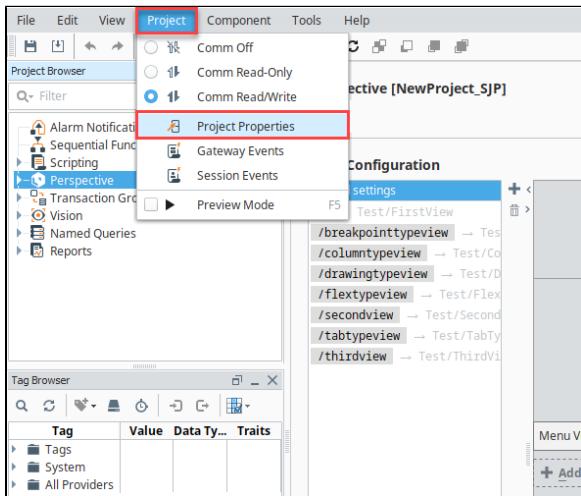
Project Properties

There are a number of properties you can set for your projects within the Designer, Vision Clients, and Perspective Sessions. For example, there are properties for setting the touchscreen mode, customizing a client's auto-login, or configuring how the clients receive updates, and many more.

The property settings on the Project Property window apply to the whole project. This page identifies and describes all the available project properties.

Accessing Project Properties

In the Designer, click on Project tab on the menu bar at the top of the Designer, then select **Project Properties**.



The Project Properties screen is displayed. Project Properties span several functional areas each containing settings applicable to that area.

Project General Properties

A project's General properties apply to the project as a whole, across all module functionality. For a new project, there are many default settings that you can use. For example, there are default settings for the Tag provider, database, publish mode, initial comm mode, window editing, and Client launching.

When properties in a section have been updated but not saved yet, the section heading will change to italicized text.

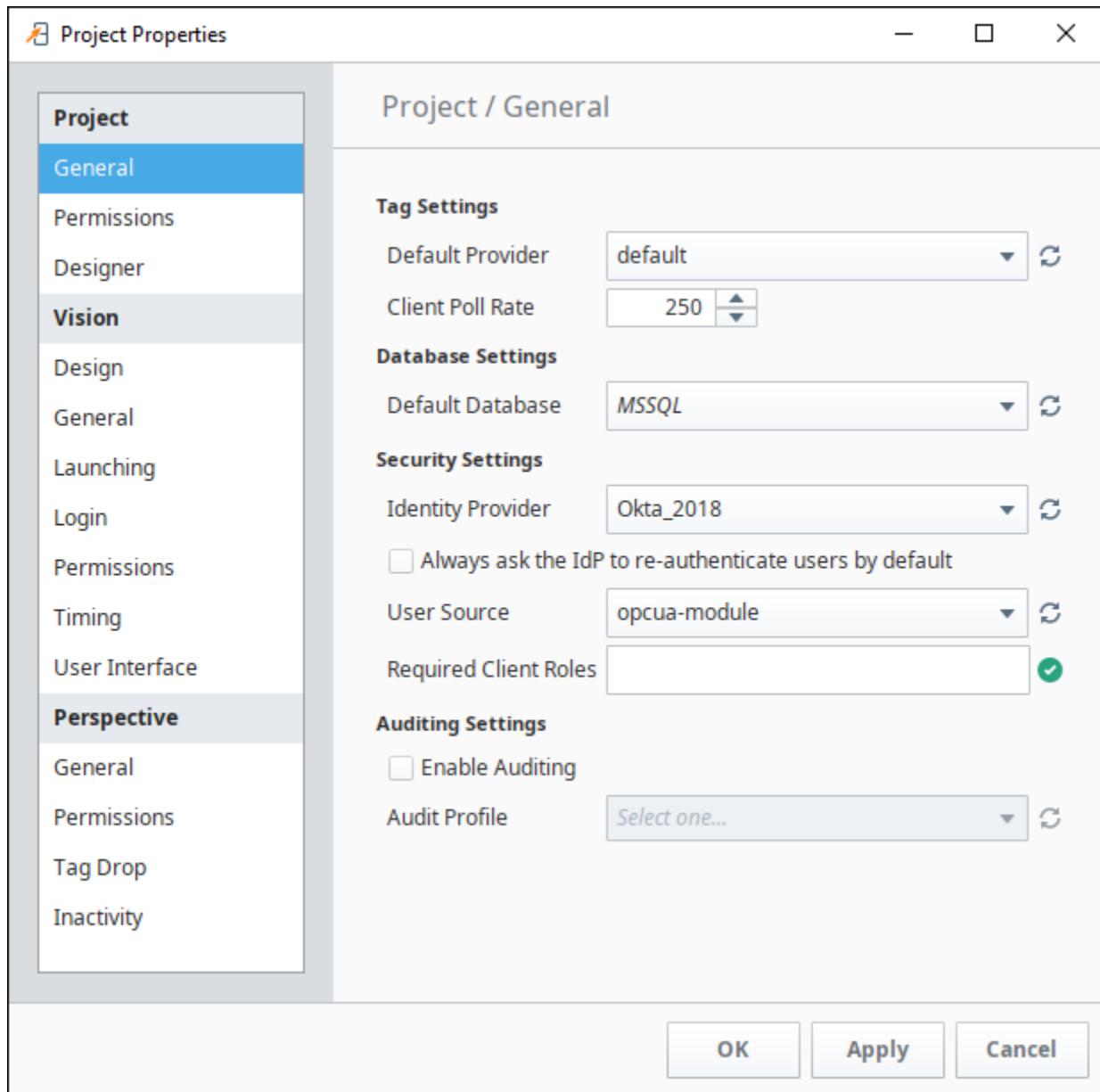
On this page ...

- [Accessing Project Properties](#)
- [Project General Properties](#)
 - [Default Database and Default Tag Provider](#)
 - [General Properties](#)
 - [Permissions Properties](#)
 - [Designer Properties](#)
- [Vision Project Properties](#)
- [Perspective Properties](#)
- [Property Inheritance](#)

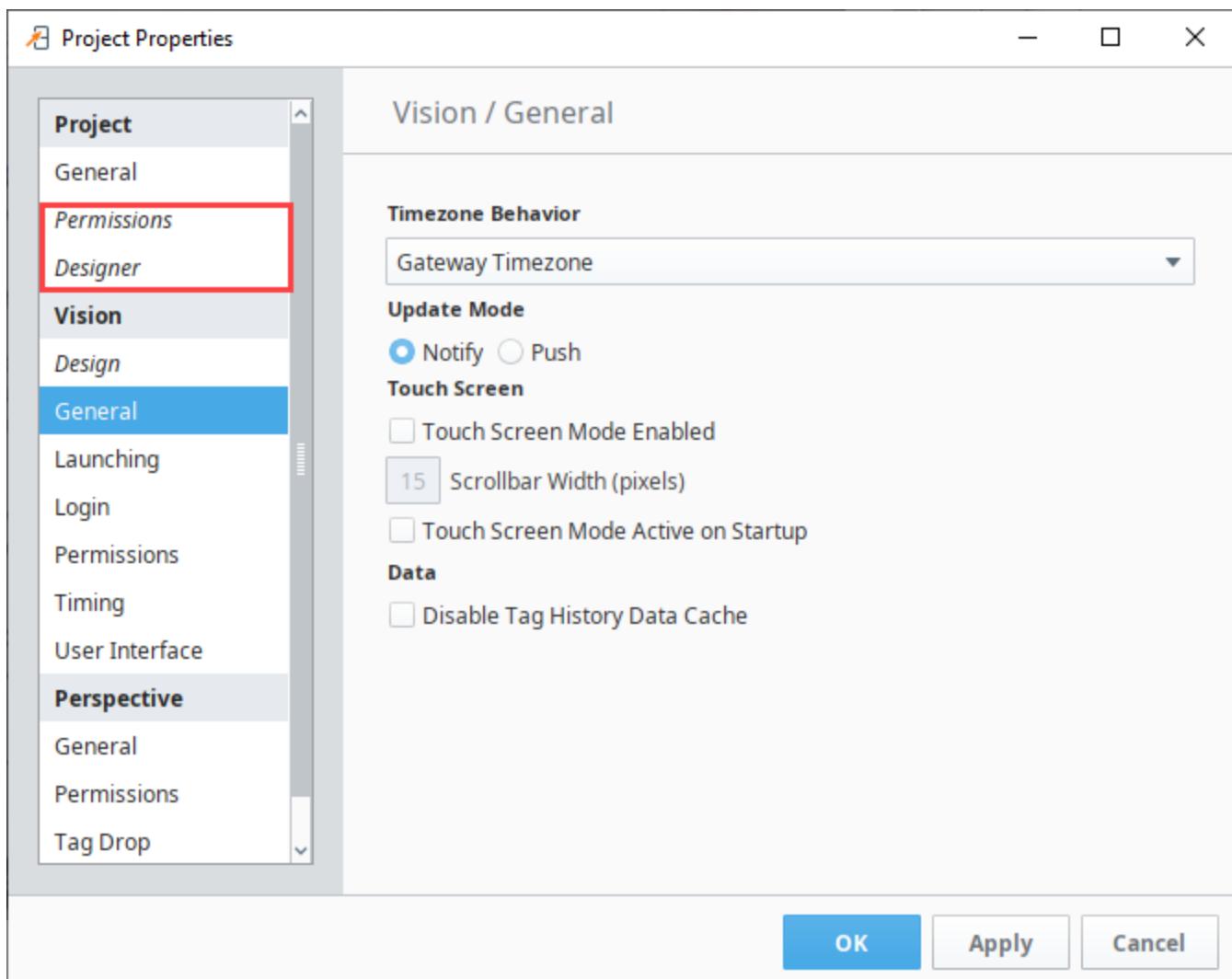


About Project Properties

[Watch the Video](#)



When properties in a section have been updated but not saved yet, the section heading will change to italicized text. In the following example, changes have been made in the the Perspective General and Permissions properties but they have not been saved or applied yet.



Note: There are a few properties of a project, such as its name, description, and title that are not available from the Designer and you need to set them in the Gateway. To do so, go to the [Gateway](#), under the **Config > Projects** section, click on the **edit** link next to the project. You cannot edit these settings while the project is open in the Designer.

Default Database and Default Tag Provider

Under **Project > General** is where you set the project's Default Database and its Default Tag Provider. It is important to understand how to use defaults effectively for proper project design.

Wherever you use a database connection or a Tag in a project, you are always given the option to use the project's default, or an explicitly named connection or provider. If your project is like most typical projects, it primarily uses a single database and a single Tag provider. By consistently using the default option, you make your project more resilient to change.

For example, suppose you have a project, and it has a database connection named `Production_DB`. Now you want to adapt the project to a new, similar plant, while leaving the existing project intact. You copy the project and create a new database connection, named `New_DB`. If your project consistently used its default database connection, the switchover will be as simple as changing the copied project's default database. However, if you used the explicit `Production_DB` connection in your groups and screens, you will need to laboriously switch the bindings over to `New_DB`.

General Properties

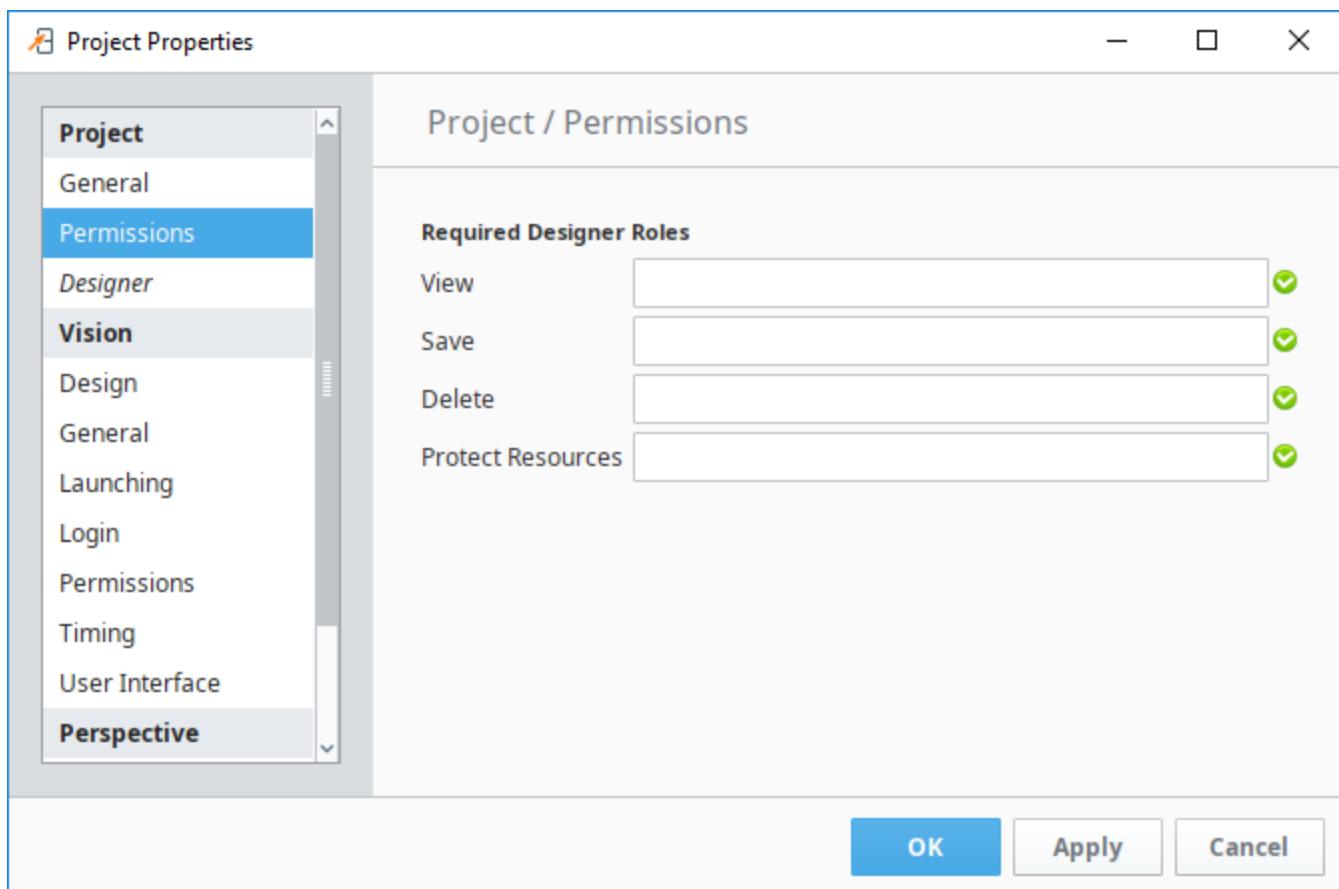
Tag Settings	
Property	Description
Default Provider	The Tag provider chosen here will act as the project's default Tag provider.

Client Poll Rate	The Client poll rate (in milliseconds) is the rate at which a Vision Client or Ignition Designer polls the Gateway for updates to its subscribed Tags.
Database Settings	
Default Database	This is the default database connection to use for this project. To use the default database connection, use the special <default> connection, or in scripting, the empty-string connection "".
Security Settings	
Identity Provider	The default Identity Provider (IdP) for this project. Choose from a dropdown list of IdPs. You can also choose <None> if the project already had an IdP but you want to remove it.
Always ask the IdP to re-authenticate users by default	When enabled, authenticated users will always need to re-enter their credentials on login. When false, if a user is already authenticated, then they will not be required to re-type their credentials when an action in the session triggers a login (such as a call to <code>system.perspective.login</code>).
<p>Note: Allowing re-authentication is entirely determined by the Identity Provider. This setting simply makes the Gateway send the re-authentication request. Consult your Identity Provider's documentation for information on re-authentication support.</p>	
User Source	Choose the User Source that governs this project's security: specifically, which group of users (User Sources) is allowed to log into the client. The User Sources are all defined in Gateway's Config section under the Security > User, Roles page.
<p>Note: This is for Vision projects only.</p>	
Required Client Roles	This property is for Client logins, and determines what role(s) a user must have before they can log into the Client. You can optionally specify a list of roles that are required for a user to log into this project. Use commas to separate the roles. Users must have at least one of the roles in order to log in. If no roles are specified, the user only needs to correctly authenticate with the User Source in order to log in. To define the roles, go to the Gateway's Config section under the Security > User, Roles page. Click the manage users link, and then go to the Roles tab. See also Security .
Audit Settings	
Enable Auditing	If auditing is enabled, audit events that relate to this project will be stored in the chosen audit profile.
Audit Profile	The audit profile stores the audit events when auditing is enabled.

Permissions Properties

When opening the project in the Designer, these properties determine which roles are required when making certain changes to the project. More details can be found on the [Project Security in Designer and Gateway](#) page.

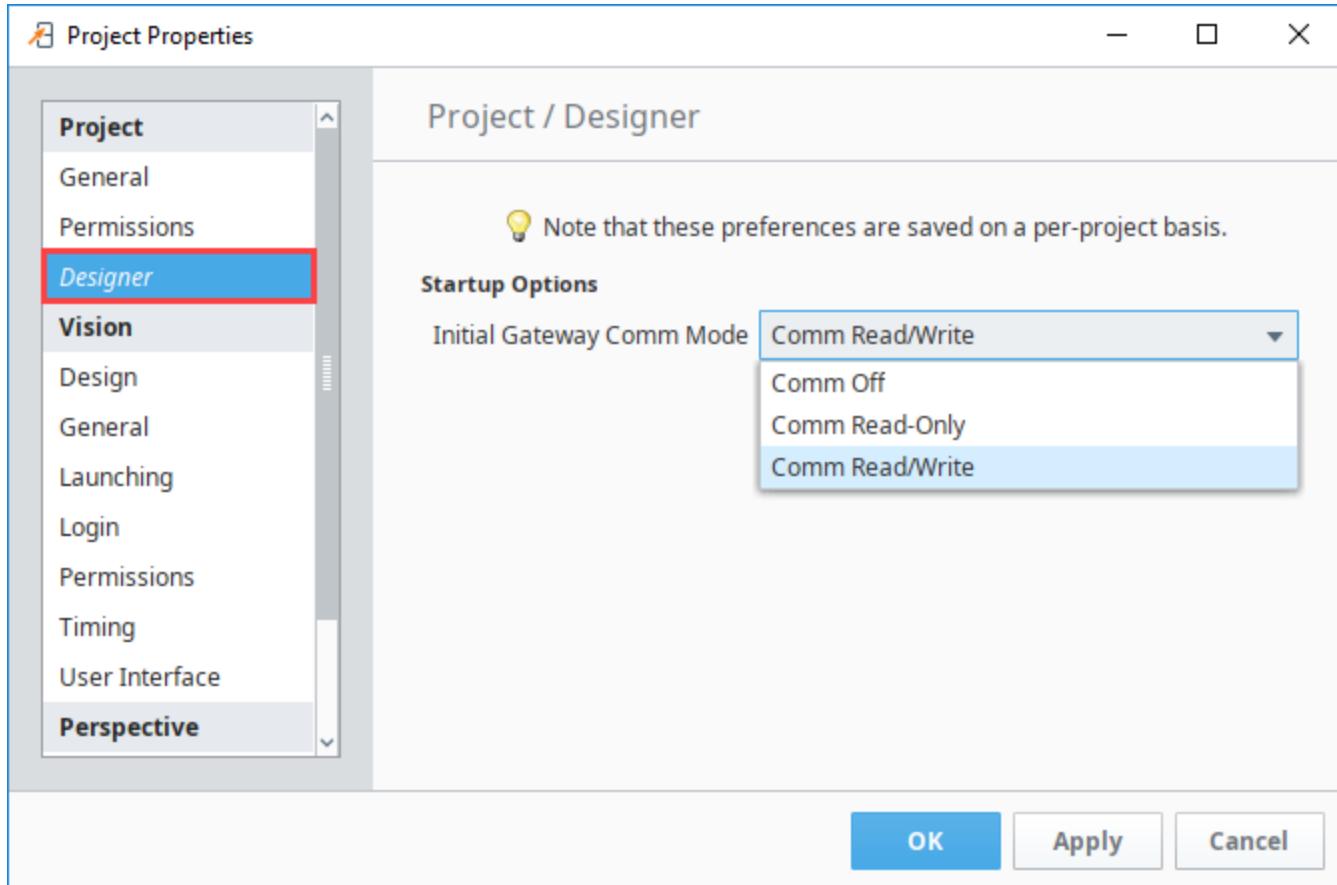
Required Designer Roles	
View	Users must have at least one of these roles to view the project in the Designer.
Save	Users must have at least one of these roles to save the project.
Delete	Users must have at least one of these roles to delete the project.
Protected Resources	Users must have at least one of these roles to access protected resources.



Designer Properties

These preferences are saved on a per-project basis.

Startup Options	
Initial Gateway Comm Mode	<p>The Designer starts up in the default Comm Read-Only mode. The property allows you to change the mode the Designer starts in when viewing the project. The options are: Comm Off, Comm Ready-Only, Comm Read/Write.</p> <ul style="list-style-type: none">• Comm Off - In this mode, all database query traffic and Tag subscriptions and writes are blocked.• Comm Read-Only - Tag subscriptions and SELECT queries work, but Tag writes and UPDATE/INSERT/DELETE queries are blocked.• Comm Read/Write - The Designer may freely request Tag and database values from the Gateway, as well as write or change these values. <p>For more information, see the Communication Modes page.</p>



Vision Project Properties

There are many project properties that apply specifically to Vision Clients. You can find more information at [Vision Project Properties](#).

Perspective Properties

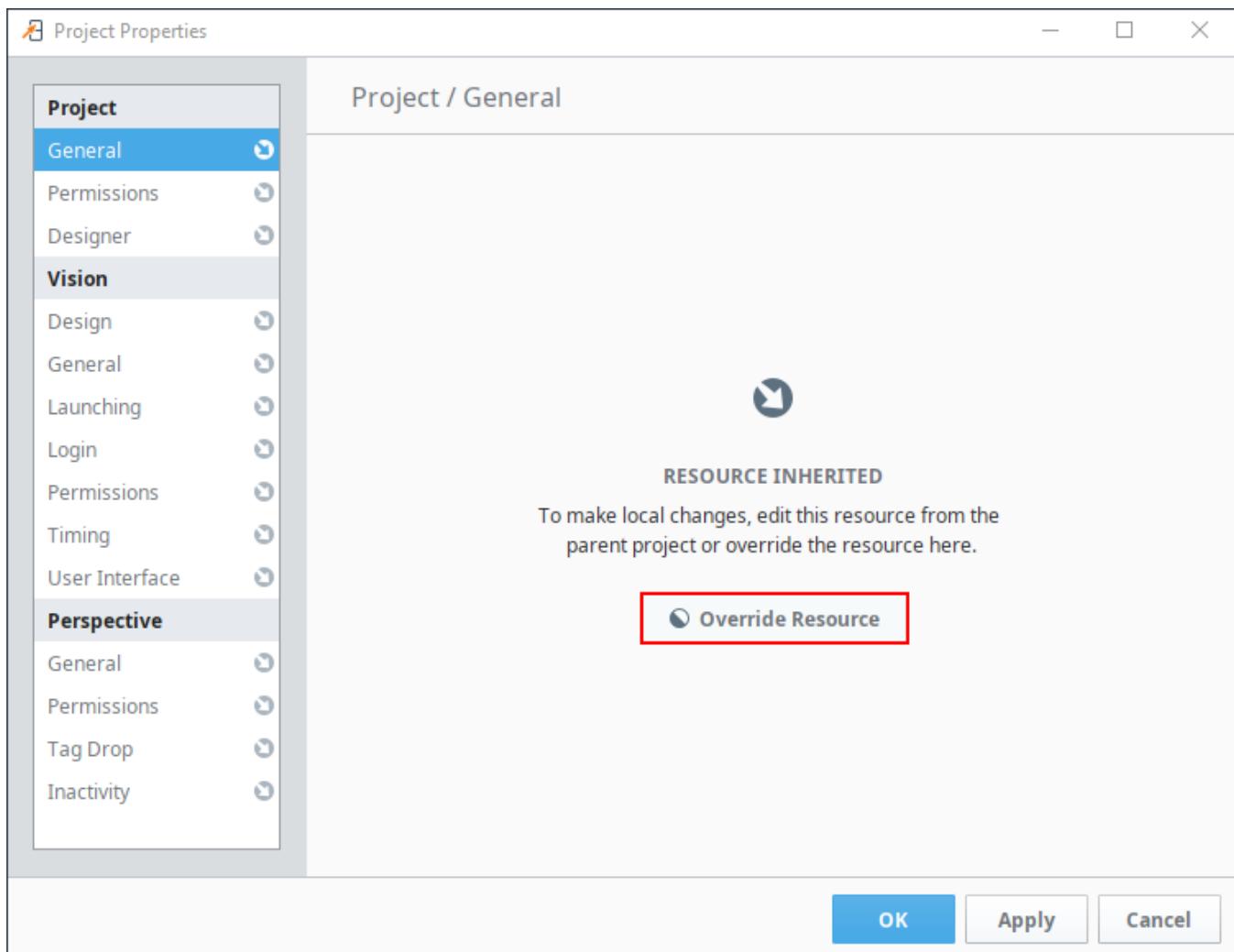
There are many project properties that apply specifically to Perspective Sessions. You can find more information at [Perspective Project Properties](#).

Property Inheritance

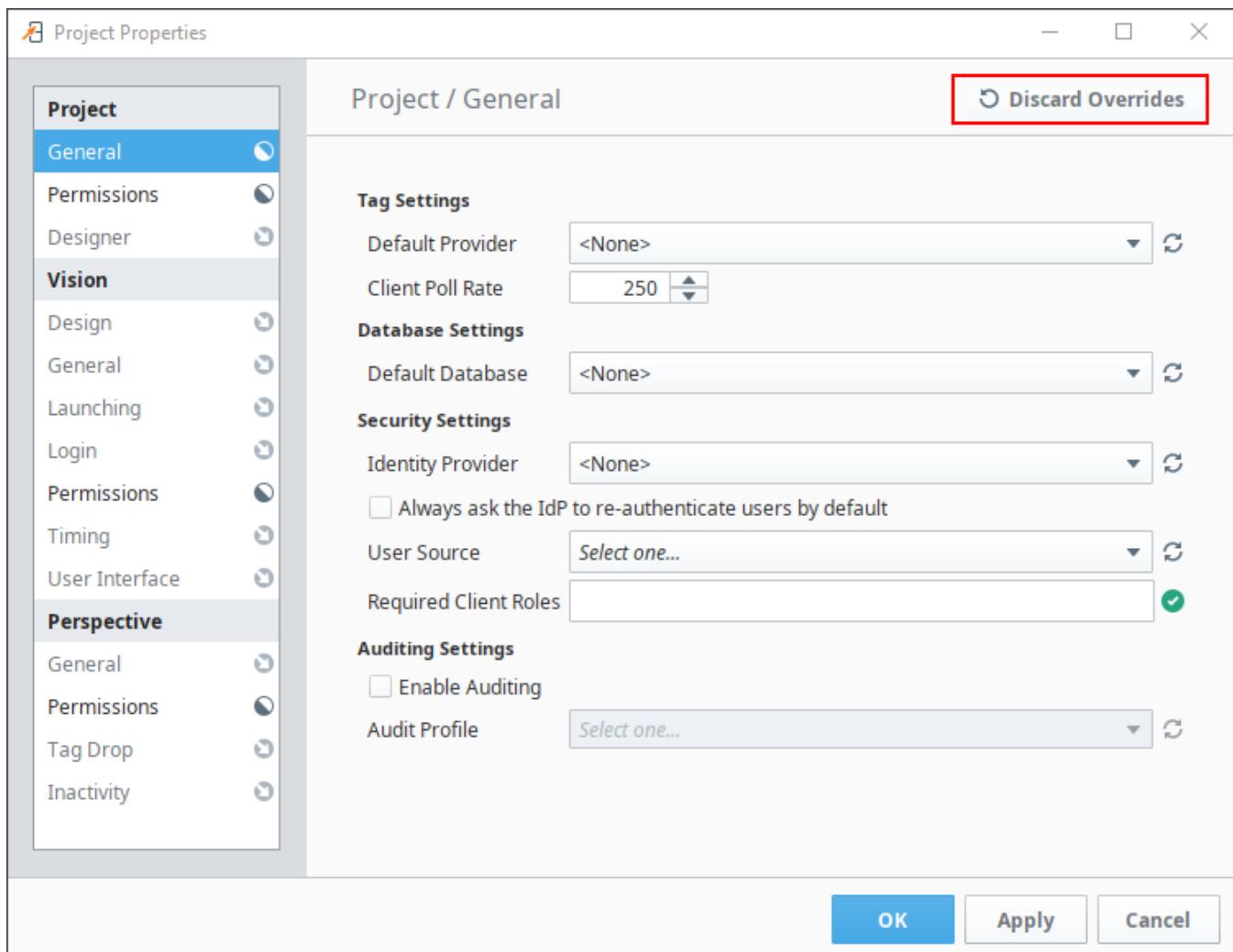
Project properties can be inherited from parent projects. You can find more information at [Project Inheritance](#).

This feature is new in Ignition version **8.1.2**
[Click here](#) to check out the other new features

Inheriting Project Properties results in a **Resource Inherited** overlay on the section. The **Override Resource** button can be used to make changes locally:

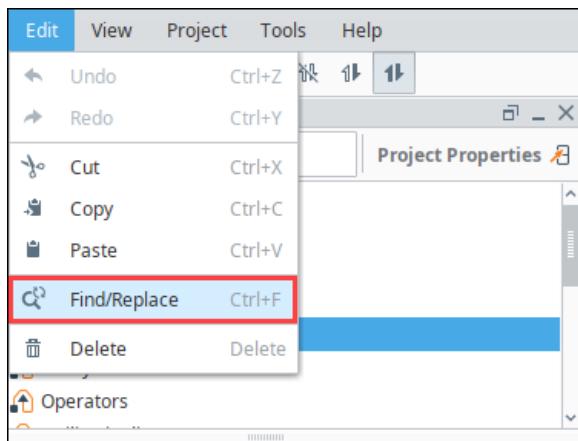


While a resource is overridden, the **Discard Overrides** button can be used to clear overrides for the current section.



Find and Replace

The Find/Replace tool in the Edit menu of the Designer allows you to search your entire project for specific components, properties, scripts, Transaction Groups, Tags, and more. You can then use the replace feature to make mass changes to a project with the click of a button.



On this page ...

- [Search Options](#)
- [Wildcards](#)
- [Using Find and Replace](#)



Using Find and Replace

[Watch the Video](#)

Search Options

In the Target section, the Find/Replace tool has options for searching through many different parts of a project.

- Pipelines
- SQL Bridge Transaction Groups
- Scripting
- Tags
- Templates (Vision)
- Views (Perspective)
- Windows (Vision)
- WebDev Resources

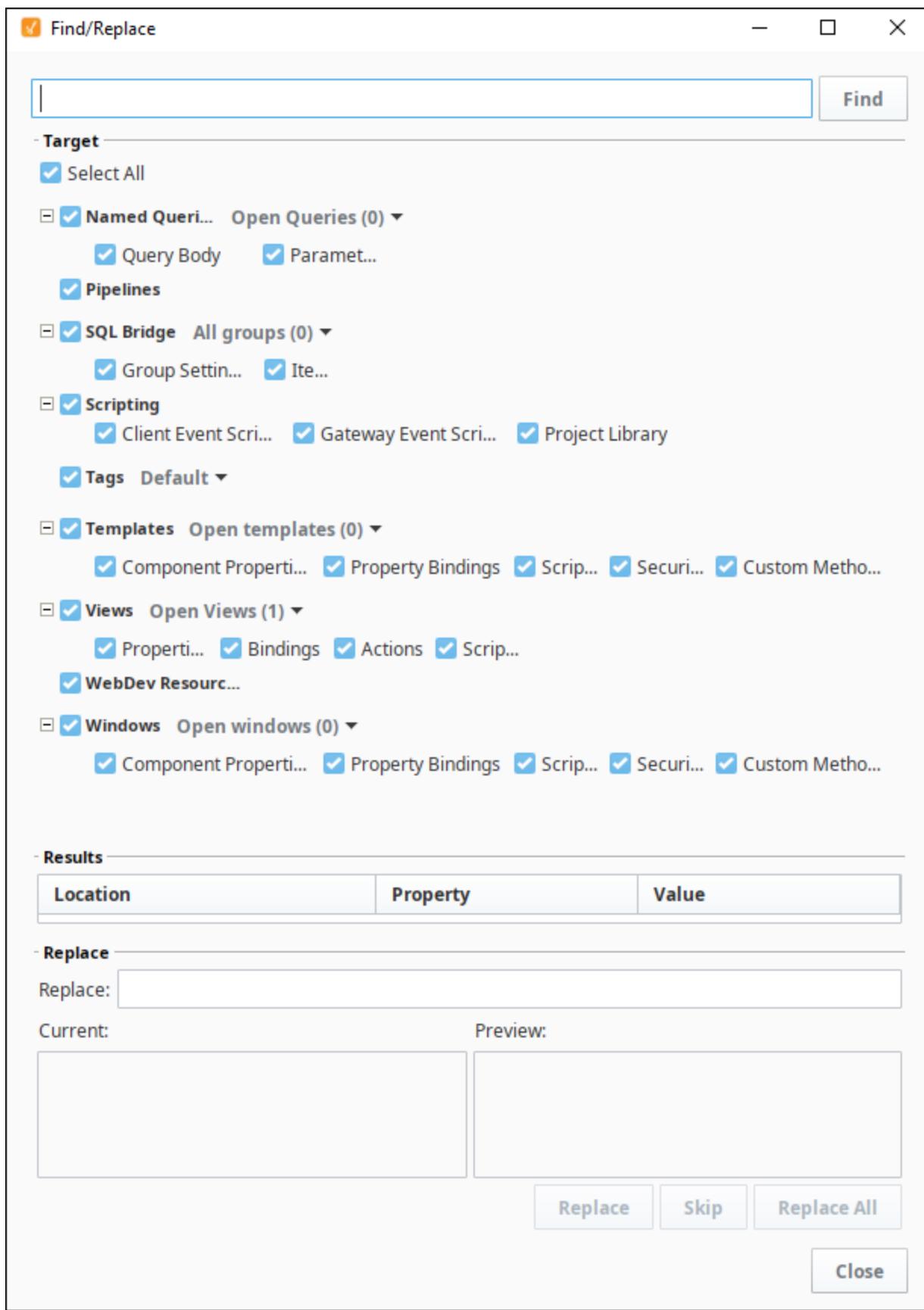
This feature is new in Ignition version **8.1.3**
[Click here](#) to check out the other new features

As of 8.1.3, The Find and Replace window can now search WebDev resources.

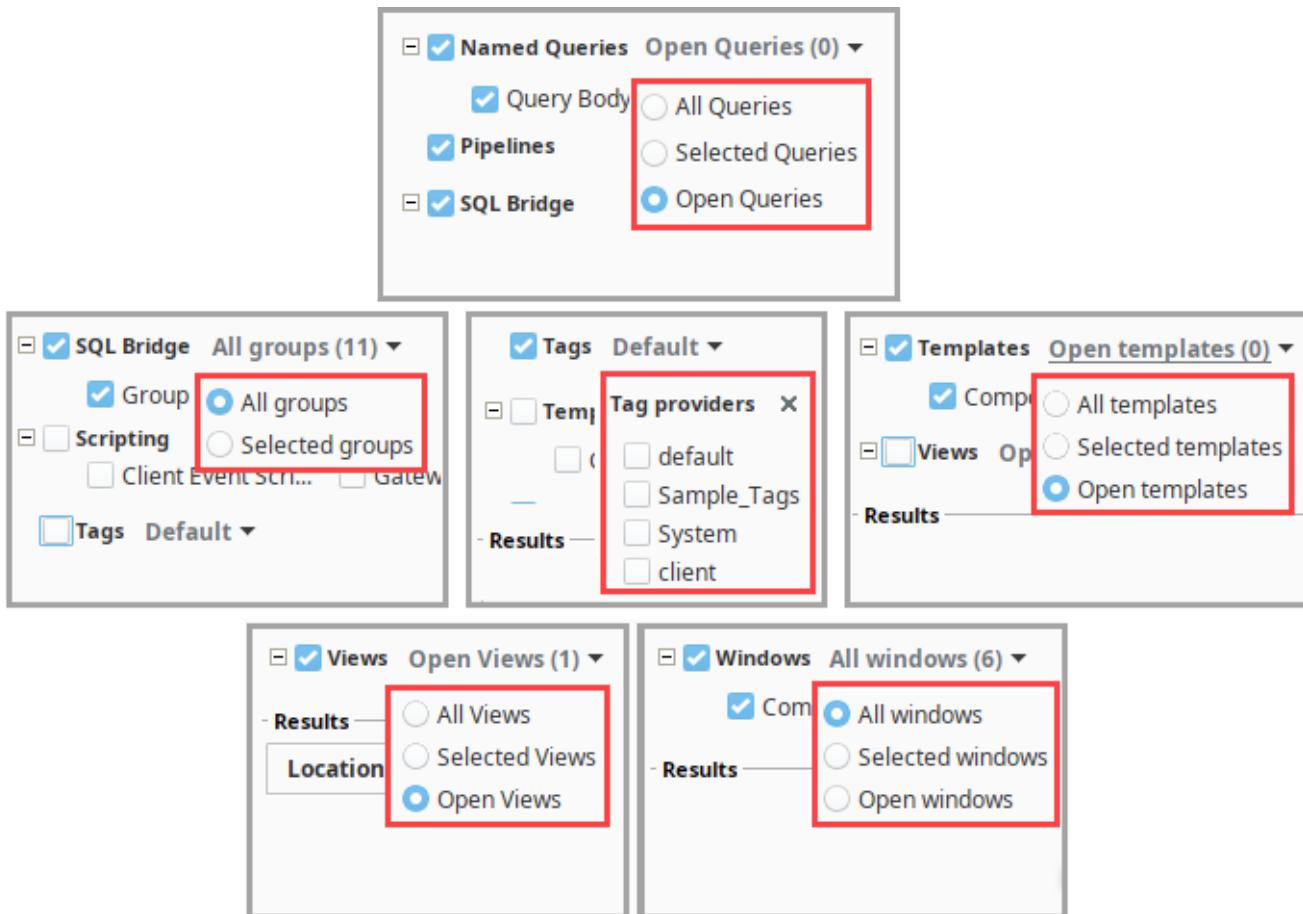
This feature is new in Ignition version **8.1.1**
[Click here](#) to check out the other new features

In 8.1.1, the Find/Replace tool added two new features:

- The Named Queries search option which lets the window search for strings in either parameters, the underlying query, or both.
- The entire tool gained a **Select All** check box, allowing you to select or unselect all possible options with a single click.

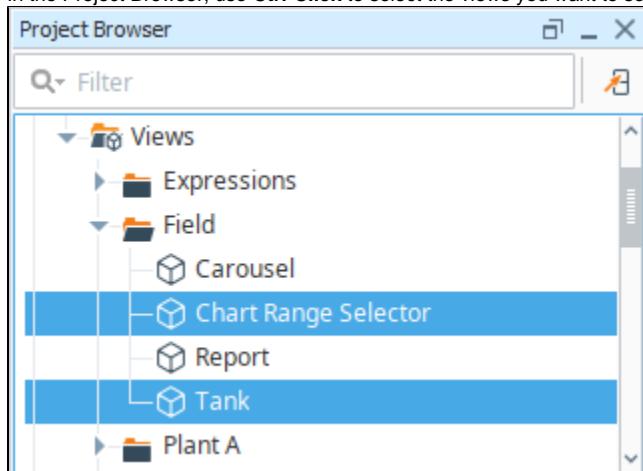


You can narrow down your search by selecting and deselecting categories you include in the search. The SQL Bridge, Tags, Templates, Views, Named Queries, and Windows options also have dropdown options to customize your search.

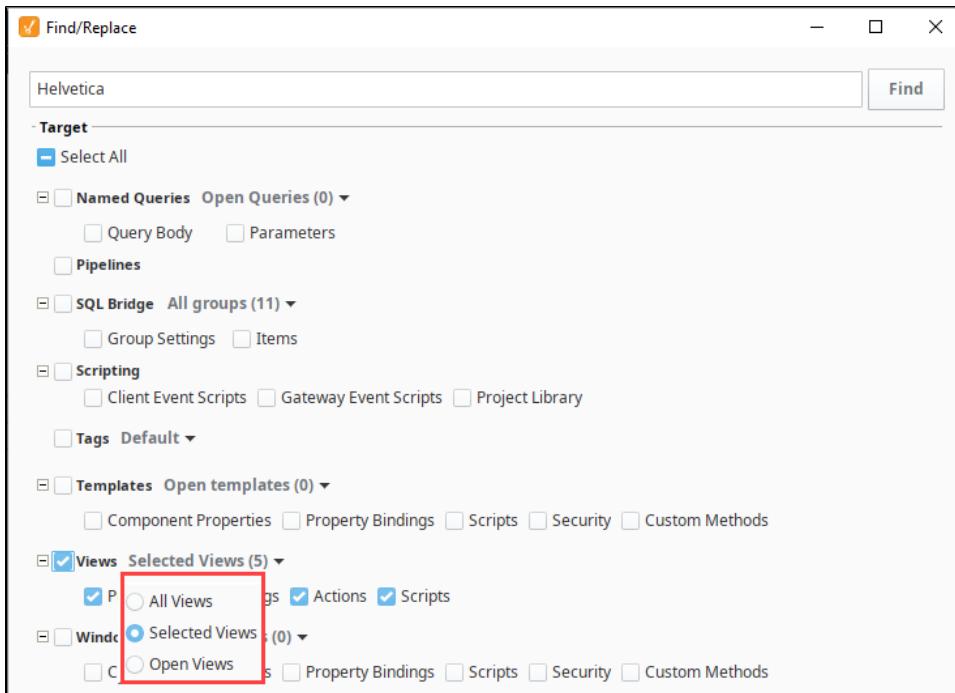


For example, if you want to search only a couple Views, do the following:

1. In the Project Browser, use **Ctrl-Click** to select the views you want to search.



2. In the Designer toolbar, go to **Edit > Find/Replace** or use the shortcut **Ctrl-F**. The Find/Replace window is displayed.
3. De-select all options by unchecking **Select All**.
4. In Views, click on the expand arrow and choose the **Selected Views** option.



5. Enter the property, action, script, or such that you want to search for. In this example, we searched for property value **Helvetica**.
6. Click **Find**.
7. The search results are displayed in the **Results** box.

Results - 4 matches found		
Location	Property	Value
views/Field/Chart Range Selector/root/ChartRangeSel..	props.style.fontFamily	Helvetica
views/Field/Chart Range Selector/root/ChartRangeSel..	props.yAxis.label.style.fontFamily	Helvetica
views/Field/Tank/root/Label	props.style.fontFamily	Helvetica
views/Field/Tank/root/Label_0	props.style.fontFamily	Helvetica

Wildcards

To customize your search further, you can use the wildcard characters * and ?.

Use an asterisk (*) to indicate that any character(s) could be where the asterisk is. For example, to search for Tags that start with the word "Motor" enter "Motor*". This would return Motor 1, Motor2, Motor_East3, and so forth.

Use a question mark (?) to indicate any single character could be where the question mark is. For example, to search for "Valve1Status" through "Valve9Status", you could enter "Valve?Status". Note that this would not return something like Valve44Status because that is more than one character where the wildcard is.

Using Find and Replace

1. In the Designer toolbar, go to **Edit > Find/Replace** or use the shortcut **Ctrl-F**. The **Find/Replace** window is displayed.
2. Enter what you're searching for in the textbox at the top to search through your project.
3. Click the **Find** button at the top right to execute the search. All matching items appear in the **Results** section. Double-click on an item in the Results table to bring that item into editing focus in the Designer.
4. To replace a value, select an entry in the Results section and you'll see the Value of that entry in the **Current** box of the **Replace** section.
5. Enter the text you want to use as a replacement in the **Replace** textbox and a preview of the new value is shown in the **Preview** box.
6. Click the **Replace** button to execute the replace. This will move your selection down in the Results table so that you can rapidly execute multiple replacements. If you're satisfied and you'd like to make the identical replacement to many items, select them all in the Results table and click the **Replace All** button.

Related Topics ...

- [Database Query Browser](#)

Keyboard Shortcuts

Using Keyboard Shortcuts in the Designer

You can interact with screens and more by using some of the popular keyboard shortcuts in the Designer.

Action	Keystrokes
Nudge Moves selected component(s) in the direction of the arrow key by the default nudge distance.	,,,
Alt-Nudge Same as Nudge, but uses the "alt-nudge" distance.	Alt + , Alt + , Alt + , Alt +
Resize Right Moves the right edge of the component left or right. Add Alt to use the alt-nudge distance.	Shift + , Shift +
Resize Bottom Moves the bottom edge of the component to top or bottom.	Shift + , Shift +
Resize Left Moves the left edge of the component left or right.	Ctrl + Shift + , Ctrl + Shift +
Resize Top Moves the top edge of the component to top or bottom.	Ctrl + Shift + , Ctrl + Shift +
Move Forward (Vision only) Moves the selected components(s) forward in the Z-order	PgUp
Move Backward (Vision only) Moves the selected component(s) backwards in the Z-order	PgDn
Move to Front (Vision only) Moves the selected component(s) to the front of the Z-order	Home
Move to Back (Vision only) Moves the selected component(s) to the back of the Z-order	End
Copy-Move Copies the component when holding Ctrl while doing a mousemove.	Ctrl + drag a component
Orthogonal-Move Restricts to only moving straight up, down, left, or right when holding Shift while doing a mousemove.	Shift + drag a component
Selection-Move Drags the components that are currently selected without having to press the mouse button down on the component first.	Hold Alt + drag a component
Copy-Axis-Move Combines copy move with axis move.	Hold Ctrl + Alt + drag a component.
Proportional Resize Resizes a component while maintaining its aspect ratio.	Hold Ctrl + resize a component
On-Center Resize Resizes the selection using the center as the anchor point.	Hold Shift+ resize a component
Select All (Vision only) Selects all components that are siblings of the selected component.	Ctrl + A
Select Same Type (Vision only) Selects all components that are siblings of the selected component and the same component type.	Ctrl + Shift + A

On this page ...

- [Using Keyboard Shortcuts in the Designer](#)



INDUCTIVE
UNIVERSITY

Using the Popular Keyboard Shortcuts

[Watch the Video](#)

Select Same Type in Window (Vision only) Selects all components in a window that are the same type as the selected component.	Ctrl + Alt + Shift + A
Layout (Vision only) Opens the Layout Constraints window to let you specify layout for the component(s).	Ctrl + L
Size & Position (Vision only) Opens Size & Position to let you specify exact size and position for selected component or window.	Ctrl + P
Customizer (Vision only) Opens the Custom Properties window to let you configure complex component properties.	Ctrl + U
Customizer 2 (Vision only) Some components have a secondary customizer. This command will open that customizer, if available.	Ctrl + 2
Jython Opens Event Configuration page (Perspective) or the Component Scripting page (Vision) to let you configure actions for component or window.	Ctrl + J
Script Configuration Opens the Script Configuration for the View or Component selected. (Perspective Only)	Ctrl + K
Security (Vision only) Opens Security Settings to let you set security for Component(s) or window.	Ctrl + E
Save Saves the project.	Ctrl + S
Open Opens the Open/Create Project to let you select a different project.	Ctrl + O
Undo Undoes the last action.	Ctrl + Z
Redo Gets rid of the last undo action.	Ctrl + Y
Copy Copies selected component(s) or window.	Ctrl + C
Duplicate Duplicates selected component(s) or window.	Ctrl + D
Cut Cuts selected component(s) or window.	Ctrl + X
Paste Pastes selected component(s) or window in clipboard. Pasted component(s) wait for position before pasting.	Ctrl + V
Immediately Paste (Vision only) Places pasted component(s) at the same location where they were copied/cut.	Ctrl + I
Comment/Uncomment Lines of Code Quickly comment or uncomment lines of a script or query in Designer.	Ctrl + /
Cancel Cancels a pending paste operation, deselects the current row of a table, cancels dragging components onto window.	Esc
Find/Replace Opens Find/Replace to let you Search and replace the project based on string, pattern, or regex.	Ctrl + F
Delete Deletes the current selection.	Delete
Snap to Grid (Vision only) Toggles whether or not moving and resizing components snaps to the grid.	Ctrl + G
Snap to Guides (Vision only) Toggles whether or not moving and resizing components snaps to guides.	Ctrl + Shift + G
Console Opens the Output Console .	Ctrl + Shift + C

Help Launches the Ignition User Manual in a web browser.	F1
Rename Renames the selected item (Tag, window, transaction group, component, and so on).	F2
Preview Mode Toggles preview/design mode.	F5
Launch Client/Session (Perspective and Vision only) When in the Vision workspace, launches a Vision Client in windowed mode. While in the Perspective workspace, launches a Perspective Session in windowed mode.	F10
Launch Full Screen Client (Perspective and Vision only) While in the Vision workspace, launches a Vision Client in full screen mode. While in the Perspective workspace, launches a Perspective Session in full screen mode.	F11
Update Project Updates project to server. Receives concurrent edits from other Designers.	Ctrl + Shift + U
Zoom Zooms in/out in the Designer.	Ctrl + Mousewheel
Touch Selection (Vision only) Draws a line while dragging. Every component in the path of the line will be selected when letting go of the mouse button. To activate, a container must first be selected	Alt + Left Click and Drag
Select Through Click on a component that is beneath another component. You can do this multiple times if there are several layers of components.	Alt + Click
Code Folding/Unfolding Select a line of code, and this command will collapse the selection. Press again while the folded code is selected, and the code will unfold. The Script Console has a separate command for code folding.	Ctrl + .

Related Topics ...

- [Saving Projects](#)

Saving Projects

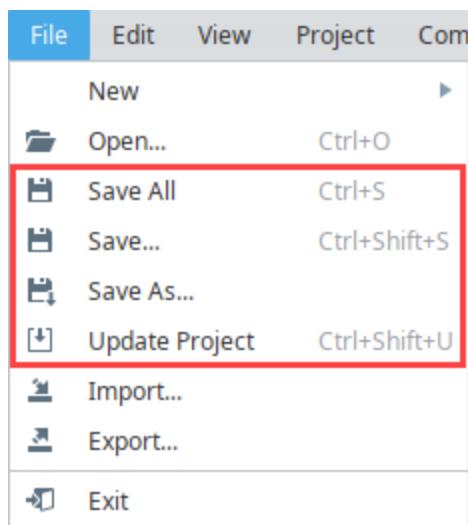
Once you created the Windows, Templates, Tags, Views, and anything else that goes into your project, you need to save your work. Saving your progress means pushing the changes from your Designer into the Gateway. If you are saving viewable resources like Vision Windows or Perspective Views, saving the changes means telling all of the clients or sessions that they can use the new updates to your project.

When you save, it's possible to save individual resources on a project as they are finished instead of saving the entire project. This is particularly helpful if you want to get resource updates to users quickly, or parts of a project into production sooner rather than waiting for the entire project to be completed.

You also have options on how to perform Client and Session updates. You can choose between notifying the operator of an available update, or automatically pushing updates as soon as it is published making it transparent to the operator. This page assumes you are using the default Notify mode, see [Client Update Modes](#) or [Session Project Updates](#) for more information.

Saving a Project

The project save options are all located in the Designer under the **File** menu.



On this page ...

- [Saving a Project](#)
 - [Save All](#)
 - [Save...](#)
 - [Save As](#)
- [Update Project](#)
 - [Project Updates in a Vision Client](#)
 - [Project Updates in a Perspective Session](#)



Project Creation and Publishing

[Watch the Video](#)

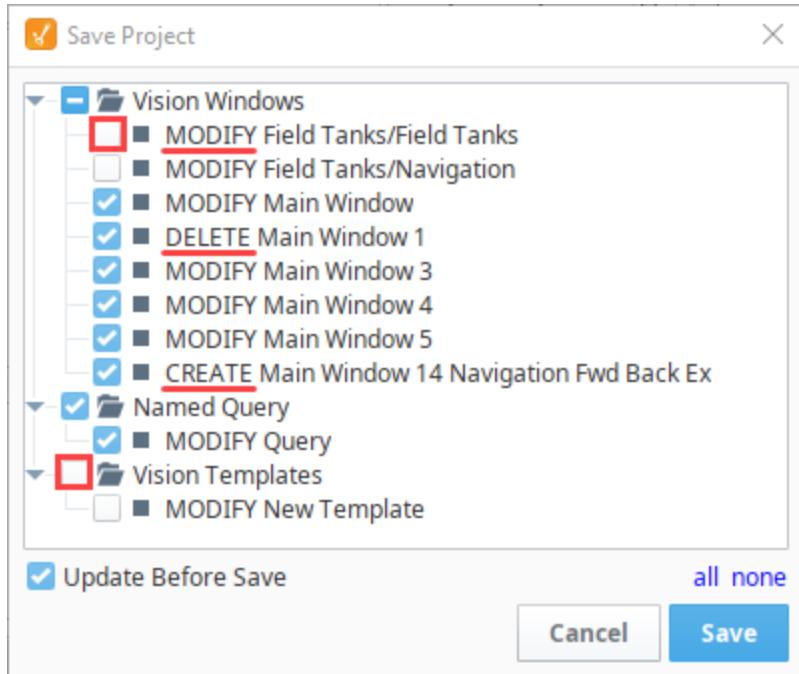
Save All

The **Save All** option saves the entire project. The project will be saved immediately, unless there are any conflicts due to [concurrent editing](#). If there are conflicts, the [Resolve Conflicts](#) screen will open automatically. Once you've resolved the conflicts, the Designer saves the project.

Save...

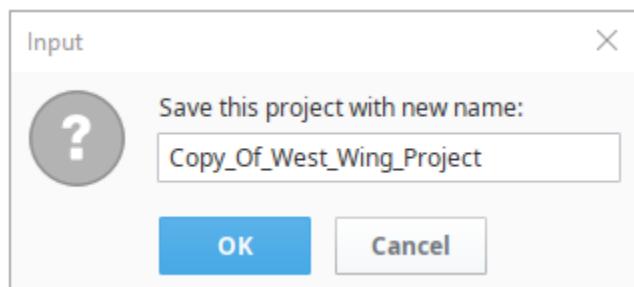
The **Save...** option displays a list of all items that have been created, modified, or deleted since the last save. On this list you can save (or not save) individual resources on a project rather than of saving the entire project.

- To deselect all items, click "none" in the lower right corner.
- To select all items, click "all" in the lower right corner.
- To select or deselect individual items, click on the Checkbox icon next to the item.



Save As

The **Save As**  option enables you to save your open project with a different name. When you choose **File > Save as**, the Designer will display an Input window. It will append "Copy_Of_" to the beginning of the current project name. However, you can enter a different name in this Input window. Click **OK** to save the project with a new name.

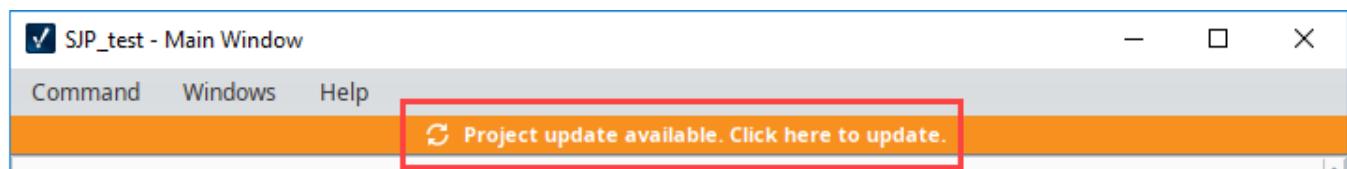


Update Project

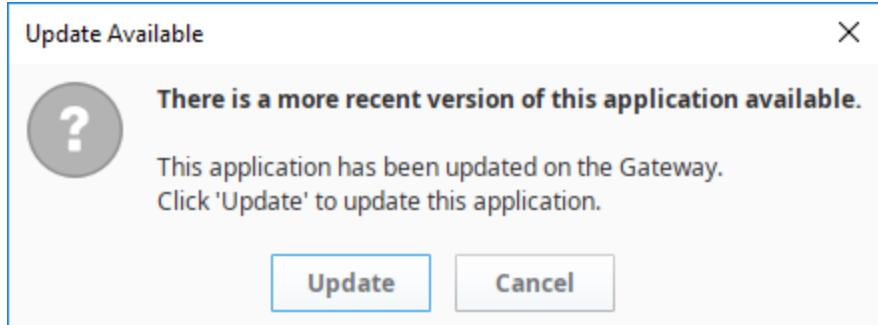
The **Update Project**  option pulls in any changes (saves by other people) that have happened since you opened the project or last updated it. If there are any conflicts due to [concurrent editing](#), the Designer [Resolve Conflicts](#) screen will open automatically. Once you resolve the conflicts, the Designer updates the project.

Project Updates in a Vision Client

Depending on how Clients are setup to receive project updates, operators may be notified with a banner stating that a project update is available. For more information, see [Client Update Modes](#).



When the operator clicks on the banner, a confirmation window appears. The operator can choose to update or cancel and keep working in the previous version of the client.



Project Updates in a Perspective Session

If you have a Perspective Session open and a change was made in the Designer that was saved and published, one of two things may happen. Either the project will silently update, or an Update Notification window will appear in the session. Your session will automatically update in 30 seconds or you can click **Update Now**. For more information, see [Session Project Updates](#).

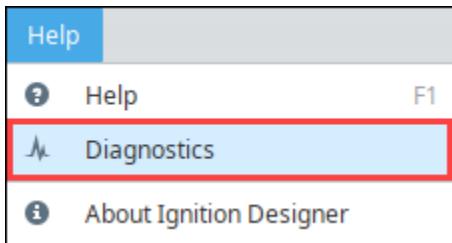
A screenshot showing an update notification in a Perspective Session. On the left is a green icon with a white downward arrow. To its right, the text 'Project Update' is followed by a timestamp 'a few seconds ago' and a collapse/expand arrow. Below this, a message reads 'This project has been changed. Please save your work, this session will automatically update in 22 seconds.' At the bottom of the notification area is a blue 'UPDATE NOW' button.

Related Topics ...

- [Vision Client Launcher](#)
- [Client Update Modes](#)
- [Session Project Updates](#)

Designer Diagnostics

The Help menu in the Designer has a **Diagnostics** option, which displays the Diagnostics window with tabs for six different troubleshooting features.



On this page ...

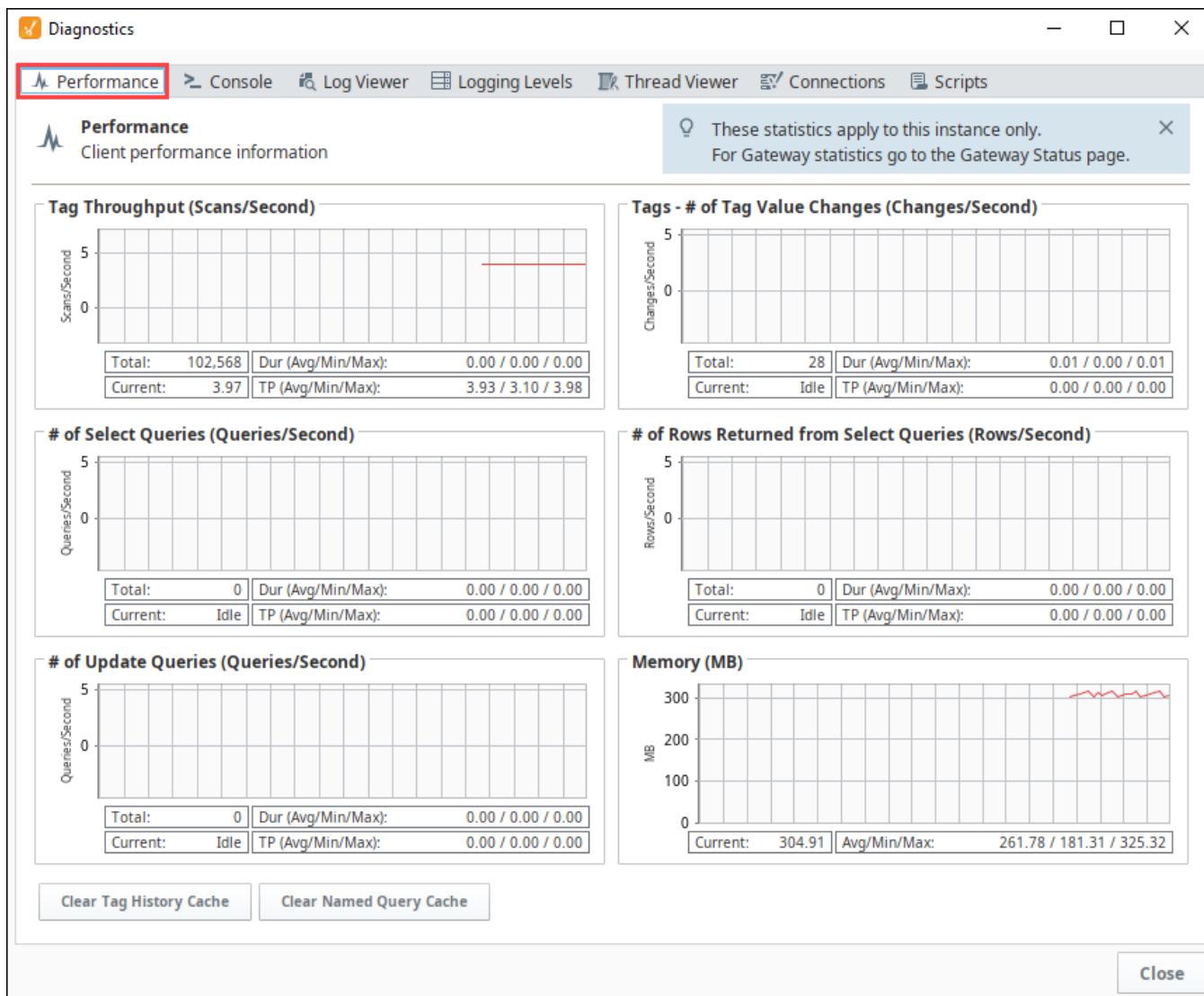
- [Performance Tab](#)
- [Console Tab](#)
- [Log Viewer](#)
- [Logging Levels](#)
- [Thread Viewer](#)
- [Connections](#)
- [Scripts](#)

Note: The information accessed through Designer Diagnostics is specific to the client runtime or the session. For Gateway statistics, see [Diagnostics - Logs](#).

Performance Tab

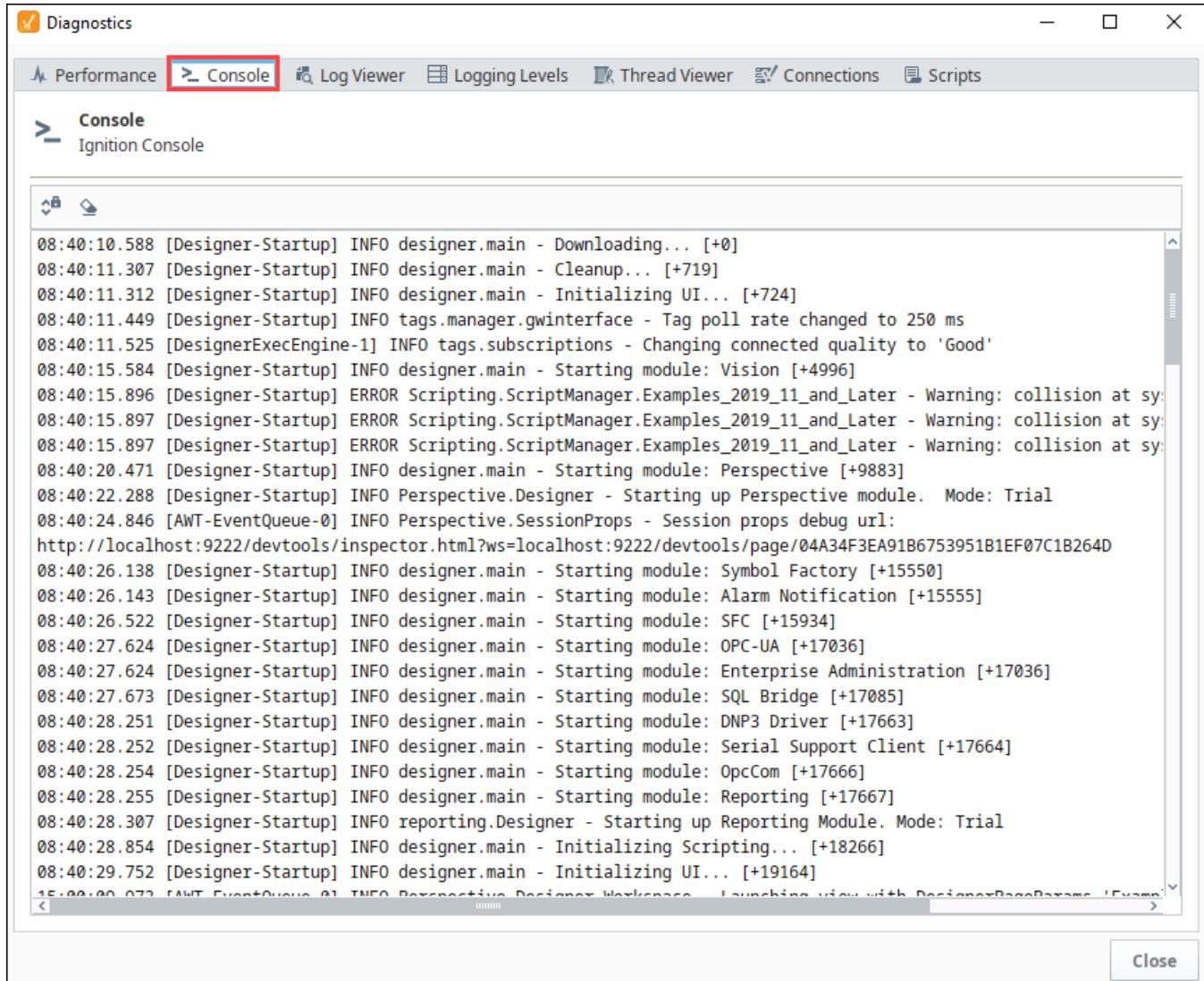
The Performance tab displays six realtime charts, each showing an aspect of the current client's performance. These charts can be very useful to help troubleshoot performance issues, especially slow queries.

- **Tag Throughput (Scans/Second)** - Displays the Tag throughput in scans per second.
- **Tags - # of Tag Value Changes (Changes/Second)** - Displays the number of Tag changes in changes per second.
- **Select Queries (Queries/Second)** - One of the most common causes of query slowdown is simply running too many queries too frequently, and the # of Select Queries (Queries/Second) chart can help identify when this is occurring.
- **Rows Returned from Select Queries (Rows/Second)** - Displays the number of rows returned from selected queries in rows per second.
- **Update Queries (Queries/Second)** - Displays the number of update queries in queries per second.
- **Memory (MB)** - Displays the client's memory usage in megabytes. This will almost always be a saw tooth pattern since memory is used, discarded, and re-acquired on a regular basis.



Console Tab

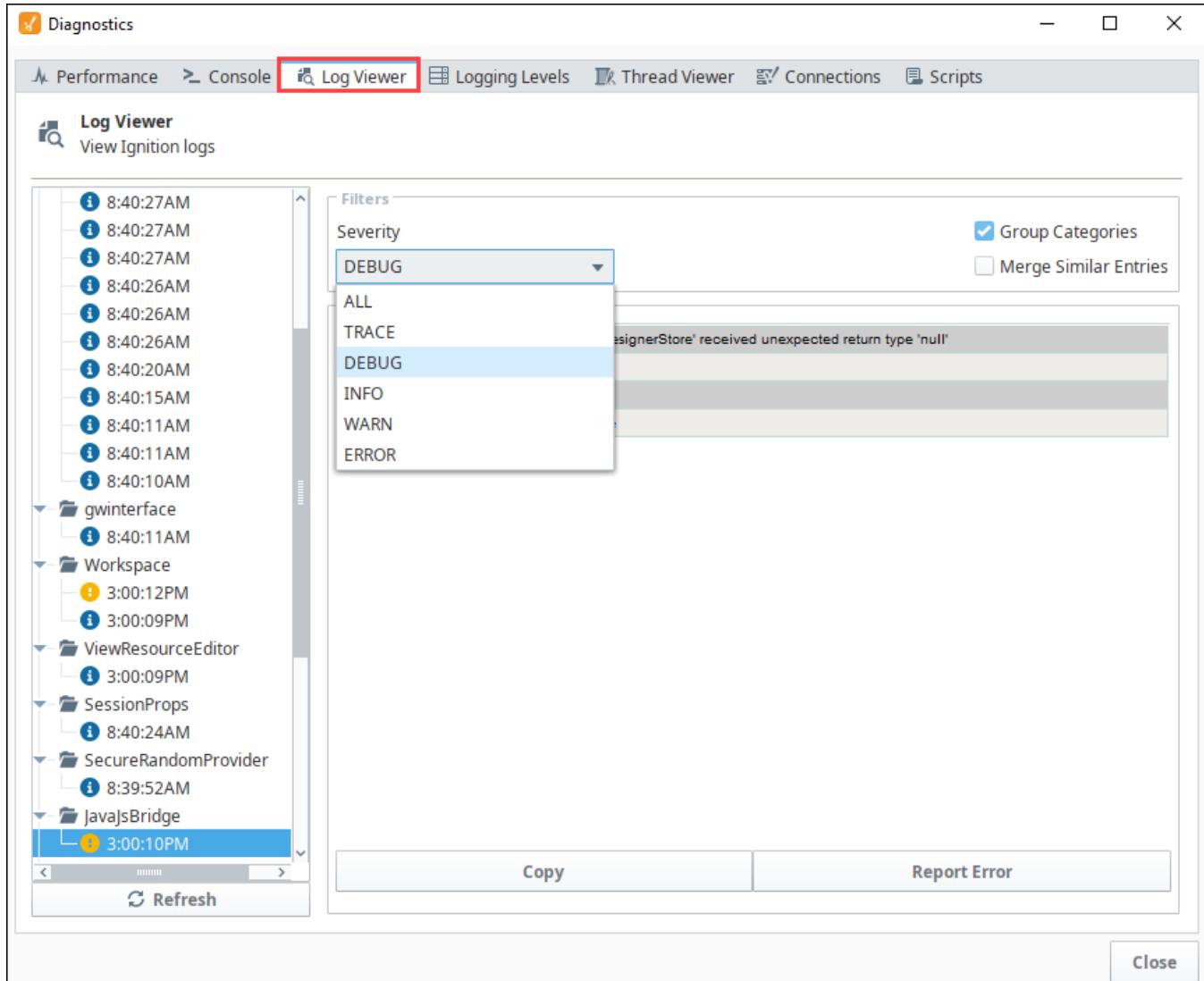
The Console tab displays the Ignition Console. This console displays messages that are generated by the entity that the console was launched from, such as the Designer. Any print statements or errors in your scripts will show up here (except Gateway scope scripts).



Log Viewer

The Log Viewer tab displays the logged events for the current entity, such as the Designer. Whenever messages occur in the console, they are logged and displayed in this tab. Each message has a logging level associated with it. This is a good place to go when troubleshooting an issue, as any errors shown here may illuminate the cause of the problem.

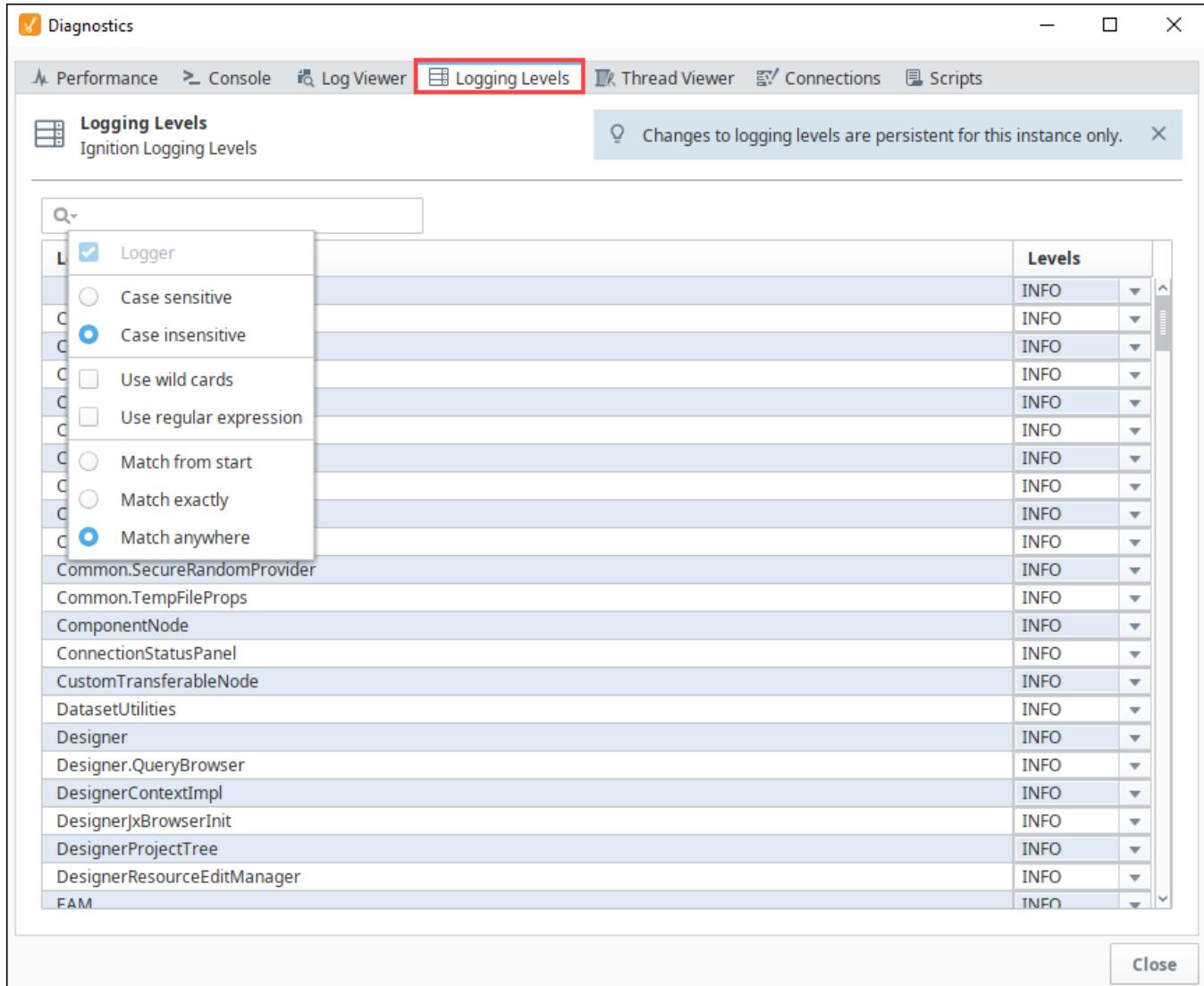
Logs can be filtered by severity by selecting an option in the Filters dropdown. To view entries across all categories chronologically, uncheck the **Group Categories** checkbox.



Logging Levels

The Logging Levels tab displays the list of internal loggers. Most users will not use this tab unless prompted by a technical support representative. Make sure to change logging levels back to info when you are done, otherwise they will flood your console and make it difficult to find any new problems.

A Search box in the upper left enables you to search the loggers. You can also set options to make the search case sensitive, use wildcards, use regular expressions, match from start, match exactly, or match anywhere.



Thread Viewer

The Thread Viewer tab displays information about the currently running threads. Each thread can be expanded by clicking the **Expand** icon or collapsed by clicking the **Collapse** icon. Most users will not use this tab unless prompted by a technical support representative.

Diagnostics

Performance Console Log Viewer Logging Levels **Thread Viewer** Connections Scripts

Thread Viewer
Shows the state of all application threads

Thread	State
Attach Listener	RUNNABLE
AWT-EventQueue-0	RUNNABLE
AWT-Shutdown	WAITING
AWT-Windows	RUNNABLE
Batik CleanerThread java.base@11.0.7/java.lang.Object.wait(Native Method) java.base@11.0.7/java.lang.ref.ReferenceQueue.remove(Unknown Source) java.base@11.0.7/java.lang.ref.ReferenceQueue.remove(Unknown Source) org.apache.batik.util.CleanerThread.run(CleanerThread.java:106)	WAITING
Browser Thread: 57763	WAITING
Chromium Process Logger	RUNNABLE
Chromium Process Thread	RUNNABLE
Client_Id_Keepalive	TIMED_WAITING
ClientProgressManager-1	WAITING
Common-Cleaner	TIMED_WAITING
designer-expr-pollingfunc-timer	WAITING
Designer-misc-1	WAITING
DesignerExecEngine-1	WAITING
DesignerExecEngine-2	WAITING
DesignerExecEngine-3	TIMED_WAITING
DesignerExecEngine-4	WAITING
DestroyJavaVM	RUNNABLE
Finalizer	WAITING
ForkJoinPool.commonPool-worker-3	WAITING
GatewayConnection-1	TIMED_WAITING
InvokeLaterTimer	WAITING
IPC Memory Reader Thread	RUNNABLE

Close

Connections

The Connections tab displays the Gateway connection status as well as a realtime chart of the Gateway ping time and a summary with the current ping time and average, minimum, and maximum ping times.

Diagnostics

Performance Console Log Viewer Logging Levels Thread Viewer Connections Scripts

Connections
Gateway Connection Status

Connected http://10.10.110.54:8088

Gateway Ping Time

Current: 0.00 Avg/Min/Max: 0.14 / 0.00 / 381.00

Close

This screenshot shows the 'Connections' tab of the Diagnostics tool. It displays a status message indicating a connection to 'http://10.10.110.54:8088'. Below this is a chart titled 'Gateway Ping Time' showing the current ping time as 0.00 ms. The chart has a red baseline at 0 ms and two small red peaks above it. At the bottom of the chart, it shows 'Current: 0.00' and 'Avg/Min/Max: 0.14 / 0.00 / 381.00'. A 'Close' button is located in the bottom right corner.

Scripts

Shows running scripts. Use the Delete icon to terminate a script.

Diagnostics

Performance Console Log Viewer Logging Levels Thread Viewer Connections Scripts

Scripts
Shows running scripts and allows termination

Close

Thread	Description	Elapsed Time
459	Vision - Button.actionPerformed	34.525s
460	Vision - Button.actionPerformed	2.961s

This screenshot shows the 'Scripts' tab of the Diagnostics tool. It displays a table of running scripts. The columns are 'Thread', 'Description', and 'Elapsed Time'. There are two entries: Thread 459 with the description 'Vision - Button.actionPerformed' and an elapsed time of 34.525s, and Thread 460 with the same description and an elapsed time of 2.961s. At the top left, there are two icons: a refresh symbol and a delete symbol. A 'Close' button is located in the bottom right corner.

Tags

What Is a Tag?

Tags are points of data and may have static values or dynamic values that come from an OPC address, an expression, or a SQL query. The values can be used on screens, in transaction groups, and more.

Tags provide a consistent data model throughout Ignition, and offer the easiest way to get up and running creating realtime status and control systems. Despite their fast initial learning curve, however, Tags offer a great amount of power in system design and configuration. The ability to aggregate Tags from a variety of installations means that you can build widely distributed SCADA systems more easily than ever before with a high level of performance and relatively easy configuration.

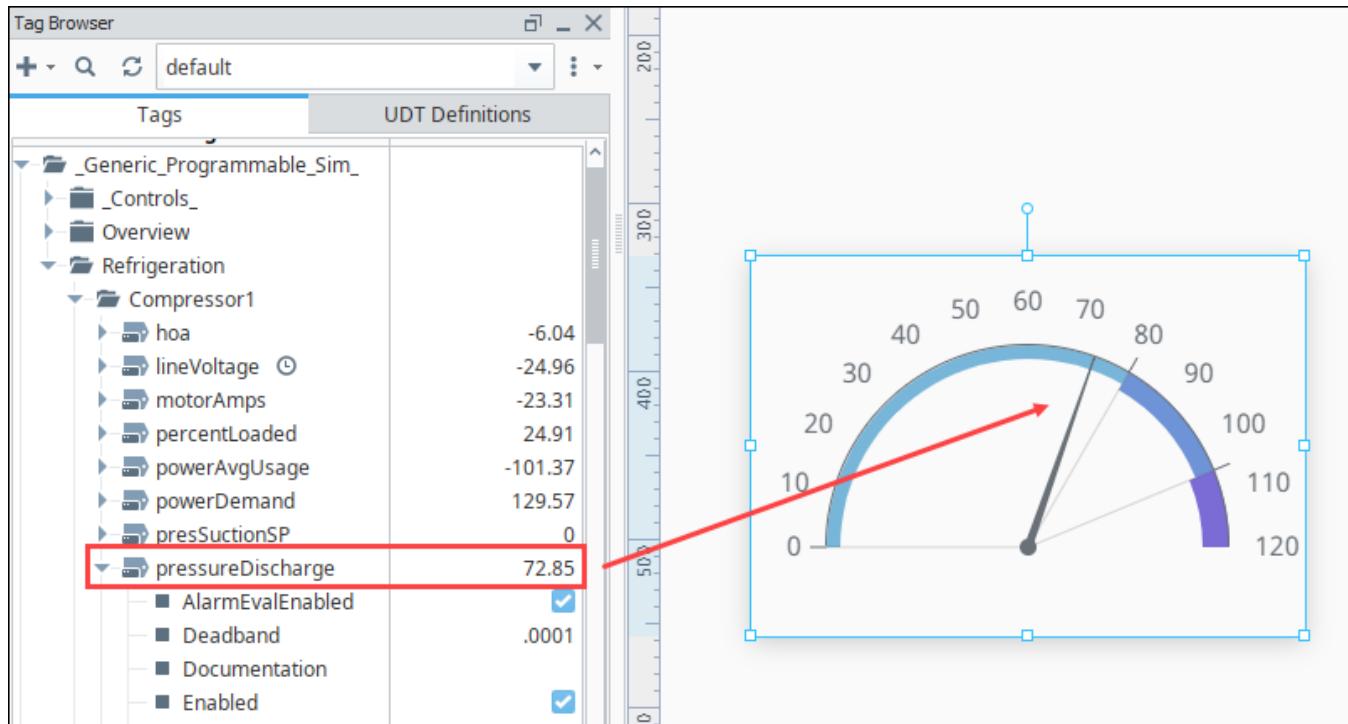
While the goal of Tags in Ignition is to create an easy yet powerful model, the variety of options and terminology can sometimes make configuration confusing. Tags are created and controlled using both the Gateway and the Designer for configuration.

- In the Designer, you create or import the Tags. There are several types of Tags such as an OPC Tags, Memory Tags, and more. Each Tag has many properties and other functionality such as alarming, history, etc. Once your Tags are created, you can use them in your windows, views, reports, and more.
- In the Gateway, you create and modify Tag Providers. You can create these Realtime Providers to store groupings of Tags for use in your projects either locally in Ignition or share them externally with connected Gateways. There are also Historian Providers used to store historical data for the Tags, but these are automatically created for each datasource you have. These Tag Provider configurations in the Gateway apply globally to all your projects.

On this page ...

- [What Is a Tag?](#)
 - [Tag Providers](#)
 - [Tag User Defined Types](#)
- [Tag Features](#)
- [Importing and Exporting Tags](#)
- [Tag Naming](#)

The following example shows a pressureDischarge Tag in the Tag Browser and a Gauge component in the Designer. The value on the Designer component is bound to the Tag and updates in realtime. This is just a simple example of how Tag values can be represented in your SCADA designs.



Tag Providers

There are two types of Tag providers; Internal and Remote. By default, a fresh Ignition installation will have an internal Tag provider. This can be thought of as a standard internal Tag database, and stored in the Ignition Gateway. Additionally, it is possible to create Remote [Tag Providers](#), linking one installation of Ignition to the Tags on another Ignition. This ability opens up some very flexible architectures.

Tag User Defined Types

Tag [User Defined Types](#) (UDTs) provide an object-oriented approach to Tag building, allowing you to define parameterized data types, extend and override types, and then rapidly generate instances. A change to the type definition is then inherited by all instances, drastically saving time when making routine changes. The UDTs are fully supported by Vision templates, which means you can configure templates for your custom data types and take advantage of drag-and-drop binding to rapidly build complex screens.

Tag Features

Tags work naturally and easily with Ignition to offer the following features:

- **Performance and Scalability**

Tags offer great performance on the Gateway, in Perspective Sessions, and in the Vision Client. On the Gateway, the system can support many thousands of value changes per second and millions of Tags. In runtime, Tags improve efficiency with their lightweight subscription architecture. Adding additional Clients creates a nearly negligible effect on the database and the Gateway performance.

- **Object-Oriented Design**

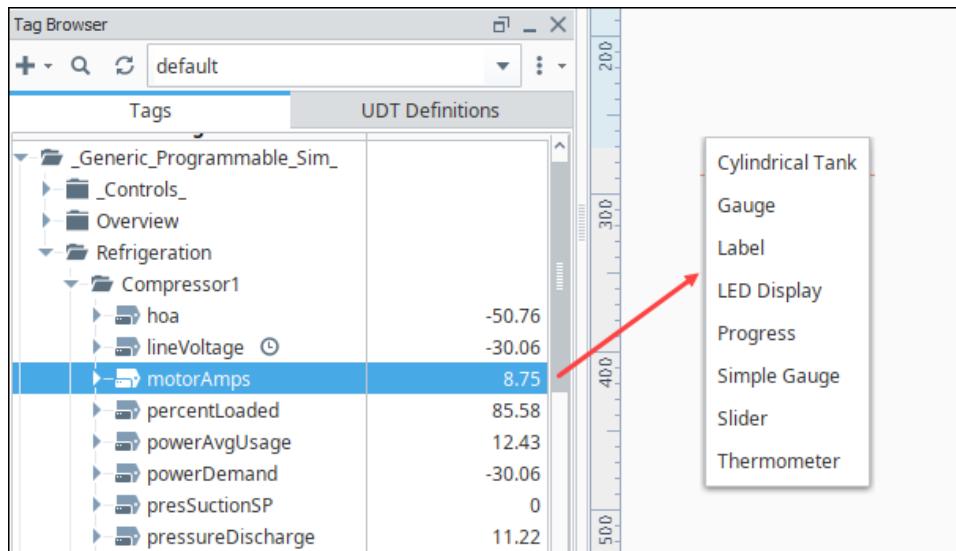
Use [Tag UDTs](#) (User Defined Types) to design re-usable, parameterized, and extendable data types. You can create and configure new instance Tags in seconds, saving a great amount of time over traditional Tag systems.

- **Powerful Alarming Model**

Each Tag can have any number of [alarms](#) configured on it. There are many different [alarm modes](#) accommodating simple digital alarms, analog high/low value alarms, as well as more specialty alarms like bad data quality and bit-packed alarms. The settings for alarms can bound to other Tags, making the alarm configuration dynamic.

- **Drag-and-Drop Screen Design**

You can drag and drop Tags onto a window or view to automatically create new bound components. Drag Tags onto existing components or properties to quickly bind them to the data.

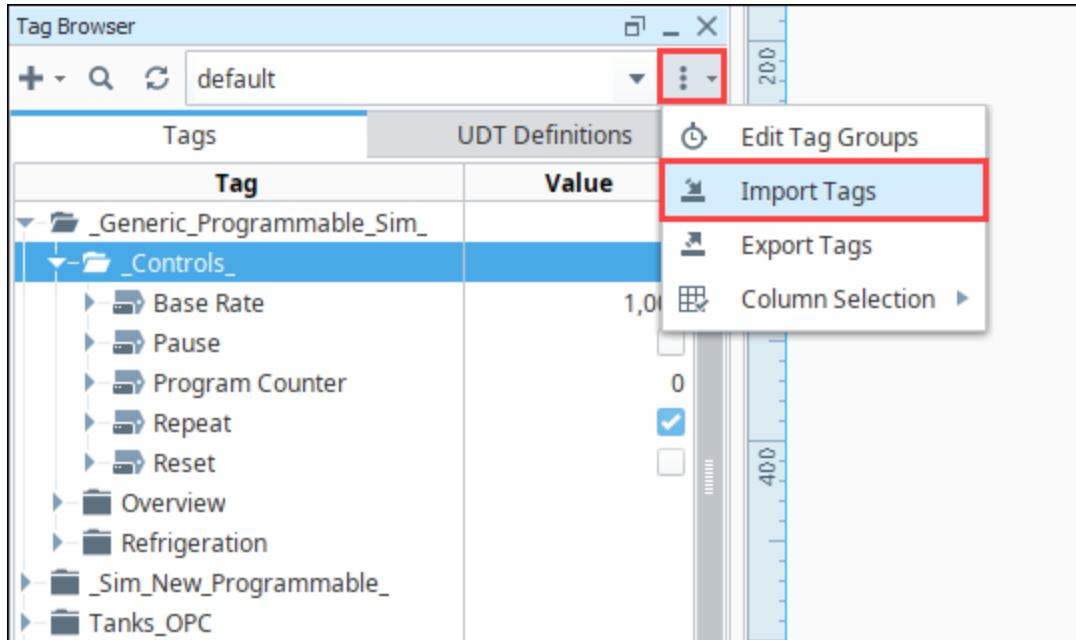


- **Historical Logging**

The [Tag Historian Module](#) makes it easier than ever to store and use [historical data](#). When you simply select a check box on a Tag, historical data is stored in an efficient format in your SQL database. This data is then available for querying through scripting, historical bindings, and reporting. Also, you can drag-and-drop Tags directly onto many components to create trends or display historical values. Tags Historian's robust querying provides you great flexibility in how you retrieve the data.

Importing and Exporting Tags

Ignition Tags can easily be imported and exported from the Designer by selecting the More Options menu, then either the Import Tags or Export Tags icon. You'll be prompted to choose the Tags or folders that you want. See the [Exporting and Importing Tags](#) page for more information.



Tag Naming

Tags names are flexible and do not have to match data source names (like an OPC path) or tag codes (such as N7, F8, etc.). It is not necessary that Tag's name be related at all to its underlying data source (OPC path, for instance). This provides a level of indirection that is convenient for systems with many repeat Tag structures.

It is important to give Tags a meaningful structure and arrange them in hierarchical Tag folders so that they are easy to understand, identify, and locate for all developers. By default, Ignition Tags are named after their OPC Server address when a Tag is dragged into the Tag Browser. You can change this name to just about anything that you want. We recommend using names that mean something to your process, such as "Motor 3 Amps." Alternatively you could create folders in your Tag Browser such as "Motor 3/Amps.". When renaming Tags and folders, there is really only one question to ask: "does this structure make sense?"

Another important concept to consider when naming and organizing your Tags, is to do this early in your project. If you rename or move any of your Tags to another folder, and your Tag is being used in other places, chances are you are going to break the reference to the Tag on your screen. So keeping your Tags organized and defining your Tag structure early on in your project is critical.

When you choose a new name for your Tags and folders, there are some rules that must be followed. The first character of the Tag name must be one of the following:

- Any alphanumeric
- Any valid unicode letter
- An underscore

The second character, and every character after that can then be one of the following:

- Any alphanumeric
- Any valid unicode letter
- An underscore
- A space
- Any of the following special characters:

' - : ()

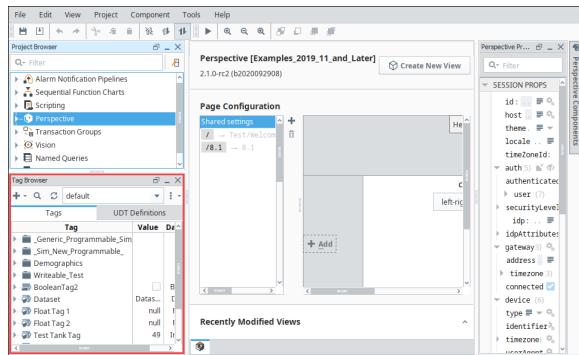
All other special characters can not be used in a Tag name.

[In This Section ...](#)

Tag Browser

The Tag Browser is the central location for interaction with all types of Tags on your system. It gives you full view of the Tags including the current value, datatype, and any traits. When panels are in their default configuration in the Designer, the Tag Browser appears on the left side.

As of release 8.1, the Tag Browser has been updated with a new design for ease of use. Tags and [UDT Definitions](#) now have their own tabs. The Tag Provider Selector enables you to view Tags for a specific Tag provider. There is a new interface for [Creating Tags](#). Many other improvements have been integrated, including icons for each [Tag type](#).



On this page ...

- [Tag Browser Tree](#)
- [Tag Browser Toolbar](#)
 - [Add Tag](#)
 - [Browse Devices](#)
 - [Find/Replace](#)
 - [Refresh Providers](#)
 - [Tag Provider Selector](#)
- [More Options Menu](#)
 - [Tag Groups](#)
 - [Import/Export](#)
 - [Column Selector](#)
- [Right-Click Menu](#)
- [Tag Traits](#)
- [Tag Editor](#)

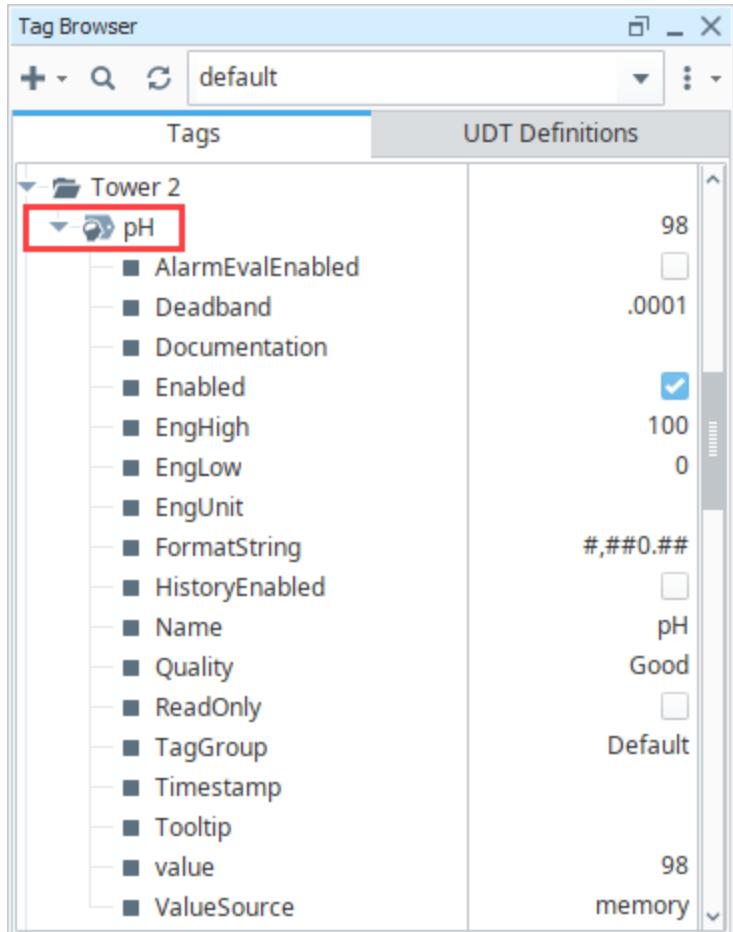


Tags in Ignition

[Watch the Video](#)

Tag Browser Tree

The Tag Browser is set up in an interactive tree structure with folders that can be expanded or collapsed to view more Tags. Click on the **Expand** ► icon to expand any folder or the **Collapse** ▼ icon to collapse the folder. In the example below, the pH Tag for Tower2 was expanded.

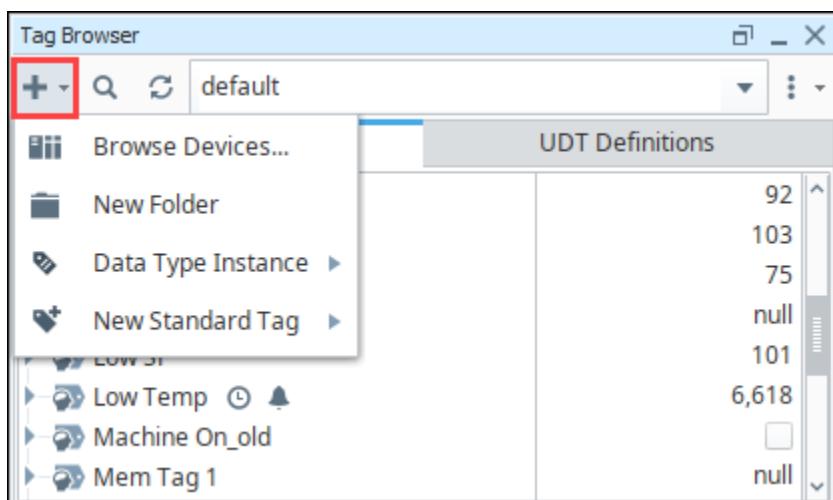


Tag Browser Toolbar

The Tag Browser toolbar has several options for working with Tags.

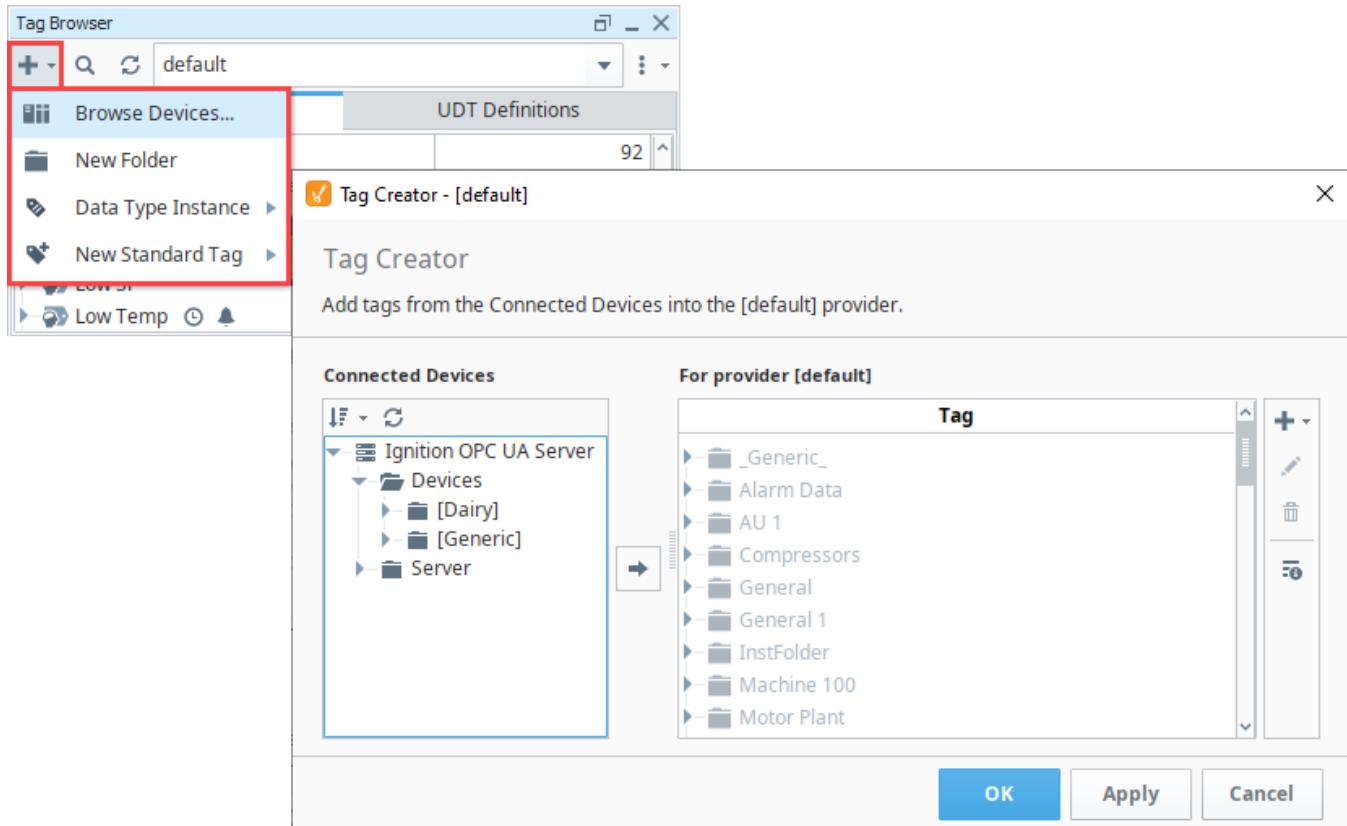
Add Tag

The **Add +** icon opens a context menu showing all the options to Browse Devices, add a Tag, Folder, UDT Instance or a UDT Definition. The new object is added under the Folder you selected, or as a sibling to the Tag you selected. This button is disabled if there is no selection.



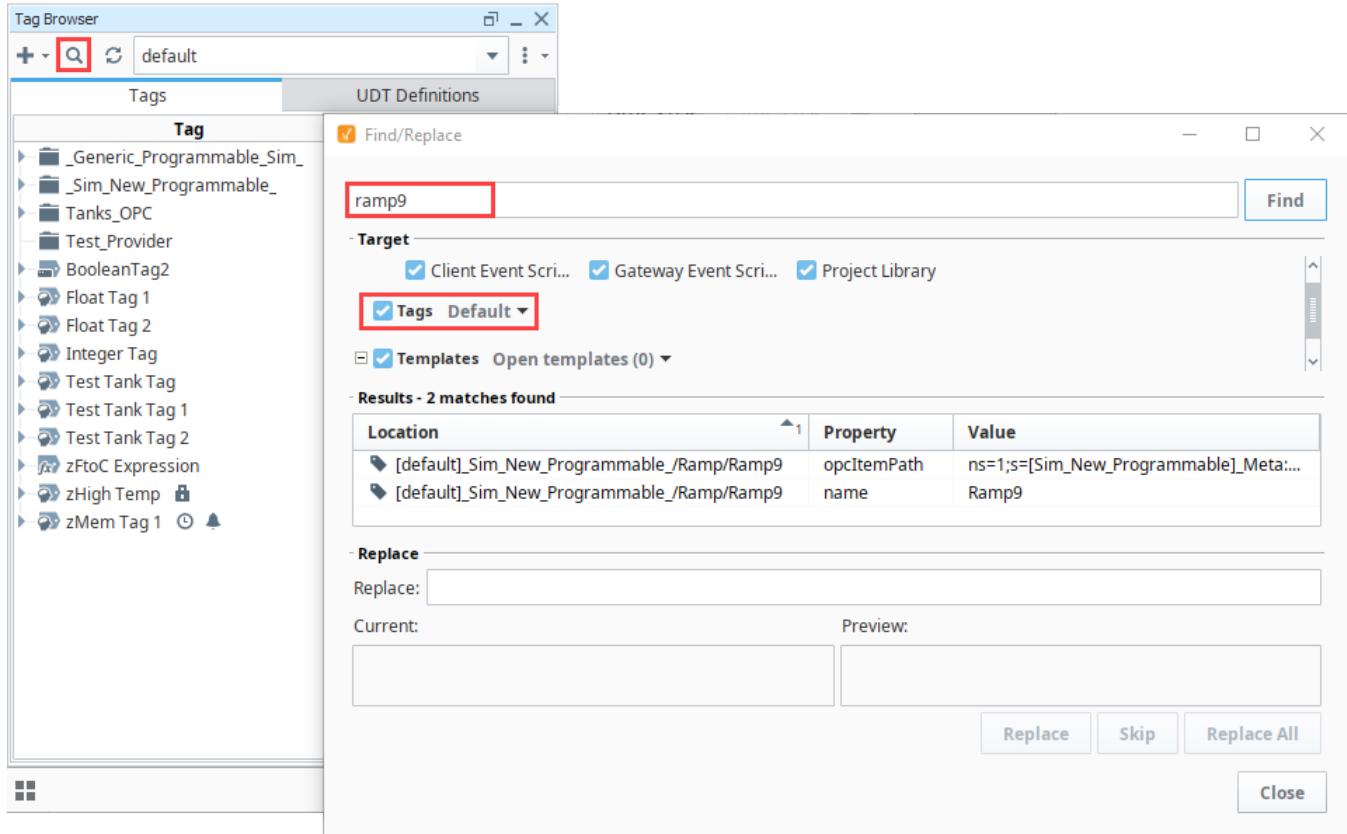
Browse Devices

With the Browse Devices, you can browse to find external PLC or OPC Tags. Click the Add  icon at the top of the Tag Browser to open the Browse Devices. You can then select Tags and move them to the Tag Browser to be used in the Ignition system. For complete information, see [Creating Tags](#).



Find/Replace

Clicking on the **Search**  icon in the Tag Browser will open up the Designer's global **Find/Replace** screen. In the example below, we searched for the Ramp9 Tag and limited the search to the default Tags. Results are shown at the bottom of the screen. For additional information, see [Find and Replace](#).



Refresh Providers

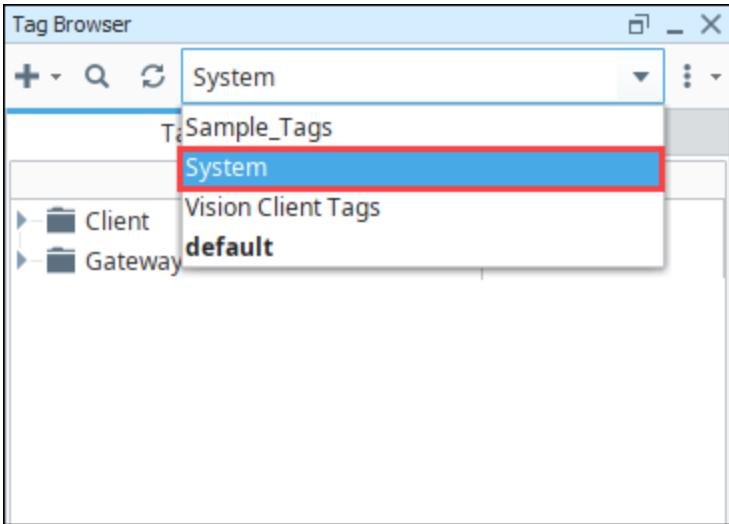
The Refresh Providers  icon refreshes all of the Providers in the Tag Browser. This is useful if you or others have modified Tags and do not see an update. In general, this button is not used very often.

Tag Provider Selector

The Tag Provider Selector is a dropdown menu with a list of available Tag Providers. Use this selector to view the System Tags, Vision Client Tags, default Tags, or other [Tag Providers](#) you have in your project.

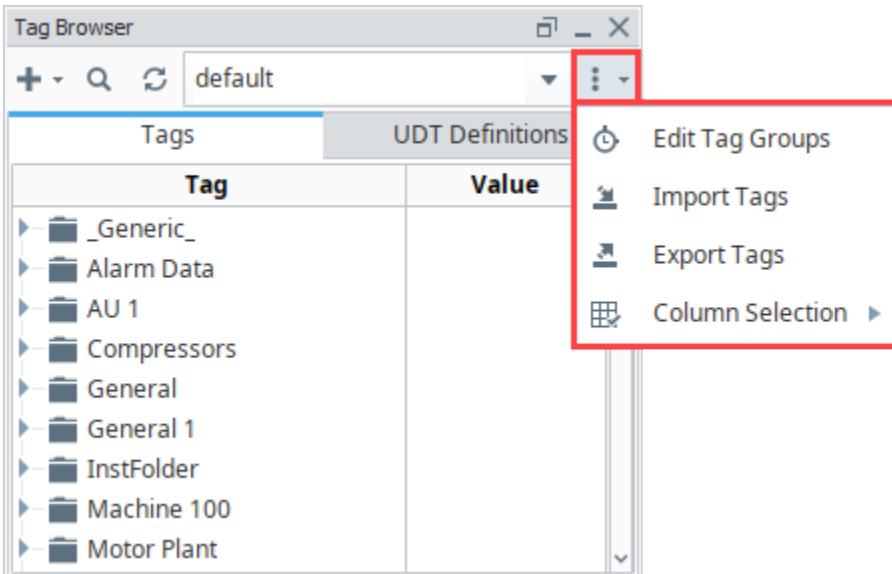
On a new install of Ignition, a single "default" Tag Provider is created for you, but there are no Tags added to the "default" provider initially. You'll notice the "default" provider is in bold. The bold entry in the dropdown list is determined by the Default Provider for the project. You can change the default provider in **Project Properties > Project > General > Tag Settings > Default Provider**. Save and restart the Designer, and the next time you open the Tag Provider in the dropdown of the Tag Browser, a different Default Provider will be displayed in bold.

When a provider is selected to **System**, the Browsing Devices under the **Add ** icon is disabled.



More Options Menu

A More options icon on the upper right of the Tag Browser, opens a dropdown menu of additional options.



Tag Groups

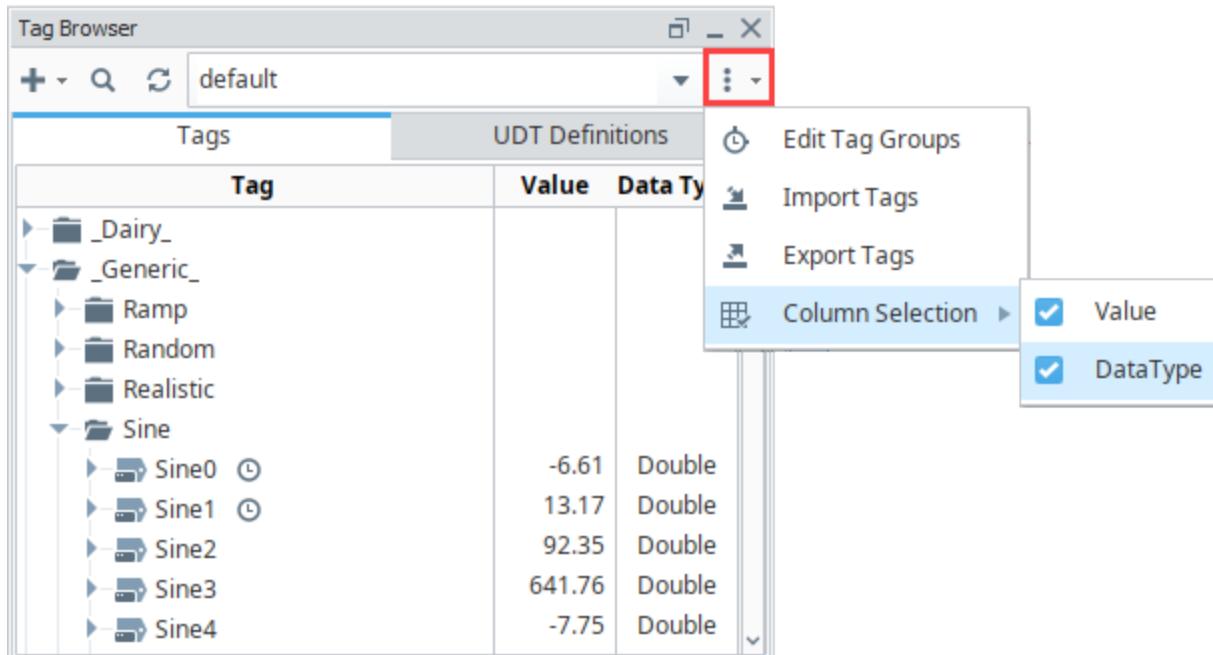
In the Tag Browser, the **Edit Tag Group** opens the **Tag Group Editor** window. Tag Groups dictate the rate of execution of Tags. This is where you set up your Tag Groups and scan rates. See [Tag Groups](#) for more information.

Import/Export

Ignition can export and import Tag configurations to and from the **JSON** (JavaScript Object Notation) file format. Use the **Import** icon or **Export** icon to import and export Tags in this Gateway. See [Exporting and Importing Tags](#) for more information.

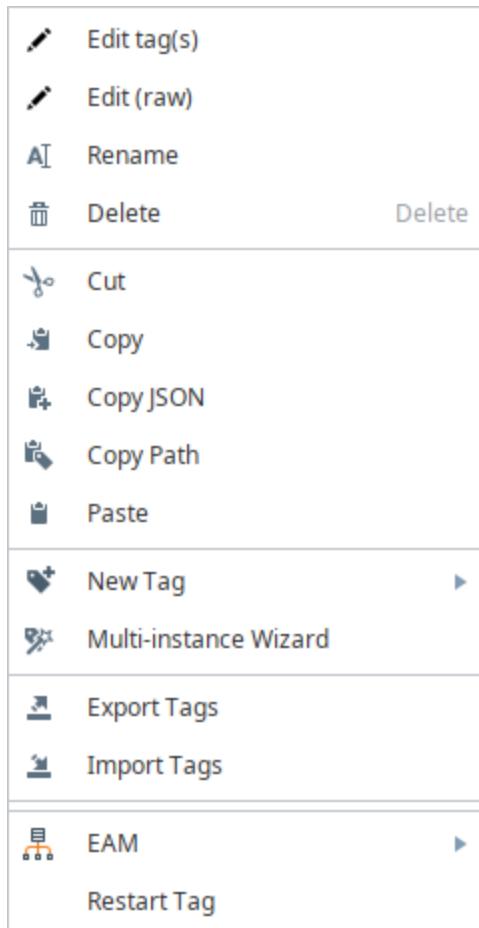
Column Selector

The Tag Browser displays the **Value** and **Data Type**. The Value type is set by default. To toggle any of the options, click on the **Column Selector** icon, then click the checkbox. In the example below, the Tag values are shown next to the Tag names.



Right-Click Menu

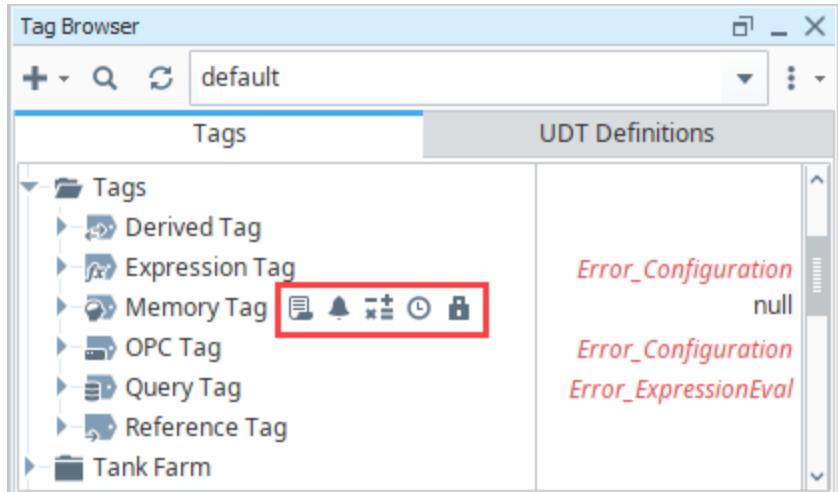
Editing Tags is done mostly through the Tag Browser. The **Tag Browser** allows you to right click on a Tag or folder to perform any of the following functions. Different objects will have different options available. The special **Data Types** folder is slightly different than a regular folder and will have even fewer options.



Function	Description								
Edit Tag	<i>Disabled when a Folder is selected.</i> Opens the Tag Editor window so the Tag can be edited.								
Edit (raw)	<i>Disabled when a Folder is selected.</i> Opens a JSON editor, allowing you to view and edit the underlying JSON that makes up the Tag.								
Rename	Renames the current selection.								
Delete	Deletes the current selection.								
Cut	Cuts the current selection into the clipboard.								
Copy	Copies the current selection into the clipboard.								
Copy JSON	Available for non-client tags only. Copies the JSON for the selected Tags into the system clipboard. In addition, pasting the JSON into a different provider/designer will create or overwrite Tags.								
Copy Tag Path	Copies the currently selected Tag path into the clipboard.								
Paste	Pastes the content in the clipboard into the selected context.								
New Tag	<i>Disabled when a Tag is selected.</i> For Folders, this option opens a sub-menu to create a Tag or Tags. <table border="1" style="margin-top: 10px;"> <thead> <tr> <th>Function</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>New Folder</td> <td>Creates a new Tag folder.</td> </tr> <tr> <td>Data Type Instance</td> <td>Creates a new instance of an existing data type. The instance is linked to the parent type so when the parent changes, the instances are overwritten with the parent type changes. Sub Menu - based on Data Types</td> </tr> <tr> <td>New Standard Tag</td> <td>Creates different types of Tags such as Derived, Expression, Memory, OPC, Query, and Reference Tags. Sub Menu - Standard Tag Types</td> </tr> </tbody> </table>	Function	Description	New Folder	Creates a new Tag folder.	Data Type Instance	Creates a new instance of an existing data type. The instance is linked to the parent type so when the parent changes, the instances are overwritten with the parent type changes. Sub Menu - based on Data Types	New Standard Tag	Creates different types of Tags such as Derived, Expression, Memory, OPC, Query, and Reference Tags. Sub Menu - Standard Tag Types
Function	Description								
New Folder	Creates a new Tag folder.								
Data Type Instance	Creates a new instance of an existing data type. The instance is linked to the parent type so when the parent changes, the instances are overwritten with the parent type changes. Sub Menu - based on Data Types								
New Standard Tag	Creates different types of Tags such as Derived, Expression, Memory, OPC, Query, and Reference Tags. Sub Menu - Standard Tag Types								
Multi-instance Wizard	Creates many instances of a UDT at the same time.								
Export	Exports the selected Tags.								
Import Tags	Imports Tags into the project.								
EAM	Sends Tag to EAM Agents.								
Restart Tag	Attempts to start / restart the selected Tag. If a folder is restarted, then all tags under the folder will restart.								

Tag Traits

Certain settings or Tag configurations are visually represented next to the Tag in the Tag Browser.



The following icons enable you to note some important settings on the Tag at a glance. A description of the icons are listed below.

Icon	Setting	Description
	Scaling	The Scale Mode property under the Numeric Tag Properties section of the Tag Editor has been set to a value other than "Off." The value on the Tag will be scaled to some degree.
	Alarming	At least one alarm has been configured on this Tag.
	Tag History	This Tag has been configured to log data into the Tag Historian system.
	Tag Event Script	At least one Tag Event Script has been enabled on this Tag.
	Lock	Shows the Tag has permissions enabled.
	Inheritance	Denotes inheritance. Displays the Parent Type, name of the UDT and instance name (i.e., Parent Types: Motor UDT > Complex Motor). Refer to UDT Inheritance Traits .
	Override	Denotes that the property in the UDT instance overrides the parent property. Refer to Overriding Properties of the Parent UDT .

Tag Editor

The Tag Editor is robust interface that contains all the properties that can be configured for Tags. In the Tag Editor, you set the Tag's name, value, numeric and meta data properties, security, alarming, history, and more. For information on all the settings in the Tag Editor, see [Tag Properties](#).

Tag Editor

Sine1

default

Properties

		▼					
Basic Properties							
Name	Sine1						
Tag Group	Default						
Enabled	true						
Value							
Value Source	OPC						
Data Type	Float						
OPC Server	Ignition OPC UA Server						
OPC Item Path	ns=1;s=[Generic]_Meta:Sine/Sine1						
Numeric Properties							
Deadband	0.0001						
Deadband Mode	Absolute						
Scale Mode	Off						
Engineering Units							
Engineering Low Limit	0.0						
Engineering High Limit	100.0						
Engineering Limit Mode	No_Clamp						
Format String	#,##0.##						
Meta Data Properties							
Tooltip							
Documentation							
Security							
Read Permissions	Public						
Read Only	false						
Write Permissions	Public						
Scripting							
Tag Event Scripts	0 event scripts						
Alarms							
Alarms	No alarms						
Alarm Eval Enabled	true						
History							
History Enabled	true						
Storage Provider							
Deadband Style	Auto						
Deadband Mode	Absolute						
Historical Deadband	0.01						
Sample Mode	On Change						
Min Time Between Samples	1						

Details

- [Documentation](#)
- [Diagnostics](#)

Min Time Units	Seconds	▼
Max Time Between Samples	0	
Max Time Units	Hours	▼▼

Related Topics ...

- [User Defined Types - UDTs](#)
- [Tag Groups](#)

Types of Tags

There are many different types of Tags in Ignition: standard Tags, [System Tags](#), and [Vision Client Tags](#). All these Tag types are available in the Tag Browser.

Tags executed in the Gateway support all of the primary features of Tags: scaling, alarming, history, and role-based permissions. These Tags run in the Gateway, and the values of the Tags are shared among all running sessions and clients. They are identical in their configurations, apart from defining how the value is generated. As of release 8.1, each Tag Type has its own icon in the Tag browser.

Icon	Tag type
	Memory Tag
	OPC Tag
	Expression Tag
	Query Tag
	Reference Tag
	Derived Tag
	System Client Tags, Vision Client Tags

On this page ...

- [OPC Tags](#)
- [Memory Tags](#)
- [Expression Tags](#)
- [Query Tags](#)
- [Reference Tags](#)
- [Derived Tags](#)
 - [Changing the Source](#)
- [User Defined Types \(UDTs\)](#)
- [System Tags](#)
- [Vision Client Tags](#)

OPC Tags

An OPC  Tag is driven by an OPC Item Path and OPC server. The OPC Item Path is a string path to a particular device connection. The exact path is defined by the driver and OPC server used to communicate with the device. Many drivers support browsing, allowing you to automatically create OPC Tags by dragging-and-dropping from the OPC Browser. However, in cases where browsing isn't supported, OPC Tags can manually be created. In the

In the **Tag Browser**, double click on any existing OPC Tag, to see the the **OPC Server** name and **Item Path**.

Tag Editor

motorAmps

default

Properties

Name	motorAmps
Tag Group	Default
Enabled	true
Value Source	OPC
Data Type	Double
OPC Server	Ignition OPC UA Server
OPC Item Path	ns=1;s=[Generic_Programmable]

Details

- Documentation
- Diagnostics

OK **Apply** **Cancel**

Memory Tags

Memory  Tags are simple Tags, that do not automatically poll or update their value. They hold the same value until some other user-created mechanism (most likely a script or binding) changes their value. They're useful in situations where a value must be stored outside of a PLC or database.

Tag Editor

Events

default

Properties

Name	Events
Tag Group	Default
Enabled	true
Value Source	Memory
Data Type	Integer
Value	35

Details

- Documentation
- Diagnostics

OK **Apply** **Cancel**

 **INDUCTIVE
UNIVERSITY**

Memory Tags

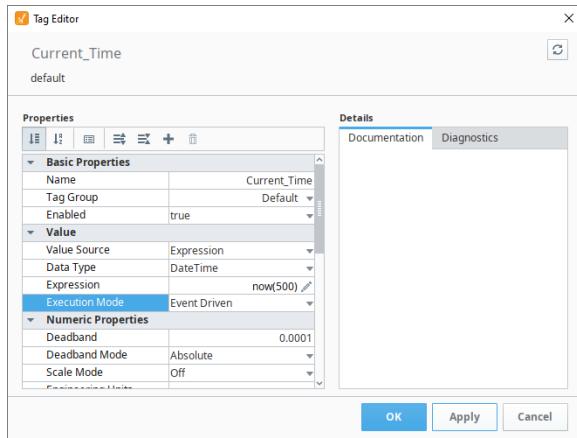
[Watch the Video](#)

Expression Tags

Expression  Tags are driven by an [expression](#), allowing their values to be determined from a calculation.

The **Expression** property on Expression Tags determines their value. The expression can reference values and properties on other Gateway scoped Tag values. However, due to scoping, they can not reference property values on Vision Client and Perspective Session components.

The expression on an Expression Tag executes based off of the **Execution Mode**. More information on Execution Mode can be found on the [Tag Properties](#) page.

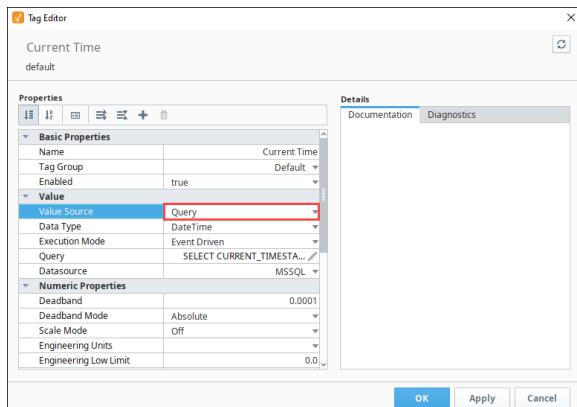


Expression Tags

[Watch the Video](#)

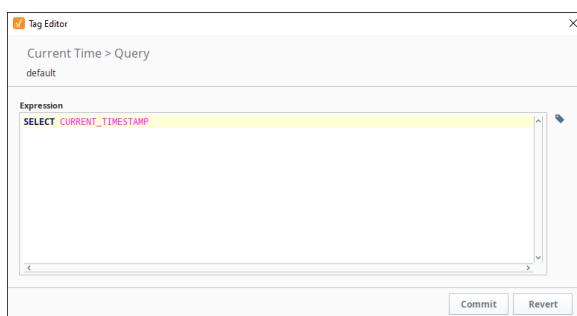
Query Tags

A Query  Tag executes a SQL Query; the result of that query is returned to the value on the Tag. Query Tags can reference other Gateway-scoped Tags to build dynamic queries. The **Query** property dictates the query that will execute, and the **Execution Mode** determines how often the query will run. Furthermore, the **Datasource** property determines which database connection the query will execute against.



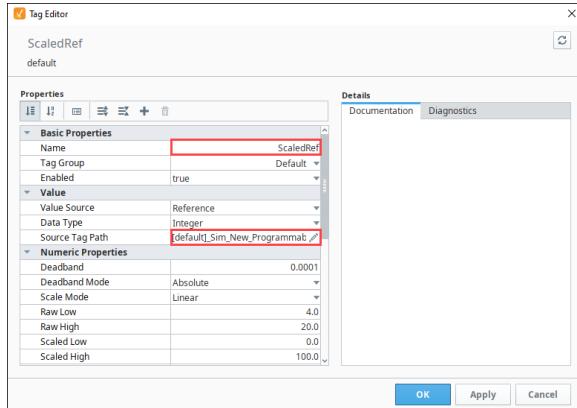
Query Tags

[Watch the Video](#)



Reference Tags

A Reference Tag simply refers to an existing Tag, using the **Source Tag Path** property to determine which other Tag to reference. Writes targeting the Reference Tag will send a write request to the source Tag.



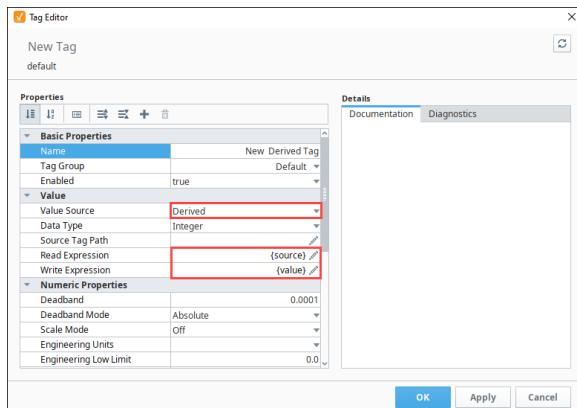
Reference Tags

[Watch the Video](#)

Derived Tags

A Derived Tag is an abstracted Tag that refers to another Tag. They are similar conceptually to Reference Tags in that that value is dependent on the **Source Tag Path** property, but Derived Tags have some additional functionality. Namely, they can apply expressions to the referenced value with the **Read Expression** property, allowing the value on the Derived Tag to differ from the source Tag.

In addition, The **Write Expression** property will apply an expression to the value of any write requests targeting the Derived Tag, allowing the expression to modify the value of the incoming write before it's applied back to the source Tag.



Derived Tags

[Watch the Video](#)

Read Expression	Determines what value should appear on the Derived Tag. The current value of the source Tag may be referenced with the {source} reference.
Write Expression	When writing to the Derived Tag, this expression determines what value should be written to the source Tag. The current value of the source Tag may be referenced with the {source} reference. The raw value passed to the Derived Tag may be referenced with the {value} reference.

This interface provides an opportunity to scale the written and read value. For example, if the source Tag was in Fahrenheit, and the derived Tag should be Celsius, we could use the following expressions:

```

//Read Expression
({source}-32)*(5/9)

//Write Expression
{value}*(9/5) + 32

```

Changing the Source

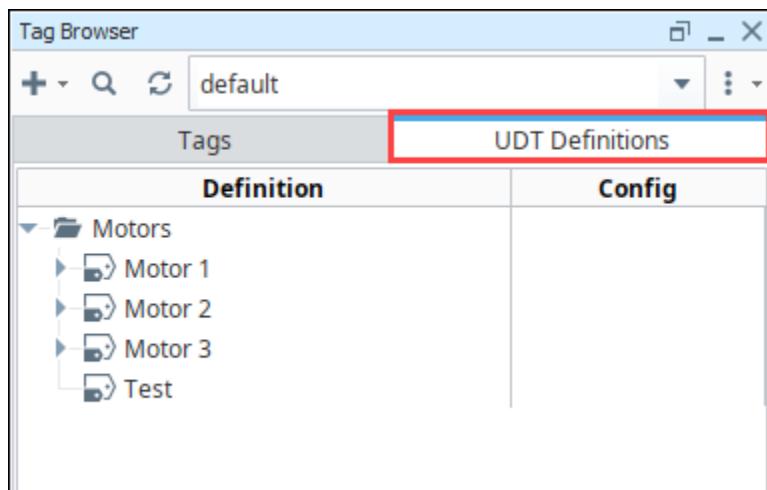
Aside from editing the Tag from the Designer, the source Tag path on a Derived Tag may be modified by writing directly to the **SourceTagPath** property on the Tag, or using a Tag Binding on a Vision component.

Additionally, the **SourceTagPath** property may be changed through scripting:

```
#Example of writing to the SourceTagPath attribute via Python Script
system.tag.writeBlocking(["Derived Example/Derived Tag.SourceTagPath"], [".]Folder/New Source Tag"])
```

User Defined Types (UDTs)

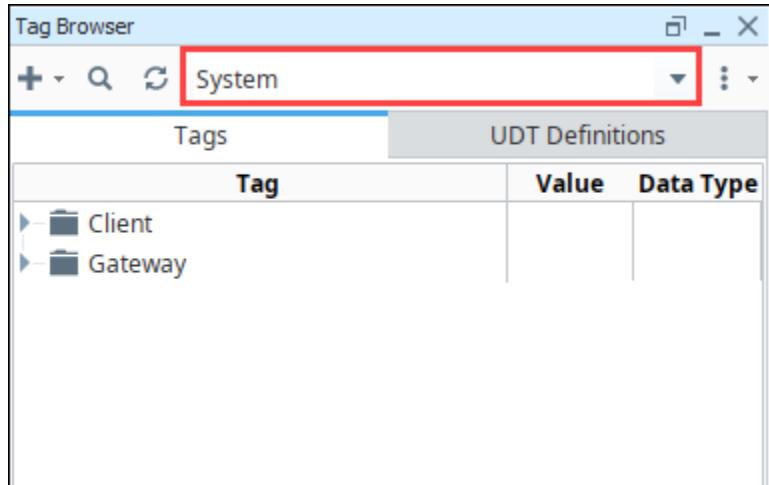
UDTs are created out of standard Tag types, but they offer a variety of additional features. You can think of them as a way to create "data templates", where a particular structure of Tags is defined, and can then be created as if it were a single Tag. This UDT example shows two Motor instances, the data type Motor, and all the Parameters and Tags that make up the structure (i.e., Amps and HOA). For more information, see [User Defined Types - UDTs](#).



System Tags

System Tags  provide status about the Ignition system. They're generally not interactable, but provide use information about how the system is performing.

More information can be found on the [System Tags](#) page.



Vision Client Tags

Within the Vision module, you can also have Vision Client  Tags that are specific to a Vision Client. Their values are isolated to a Client runtime. For more information, see [Vision Client Tags](#).

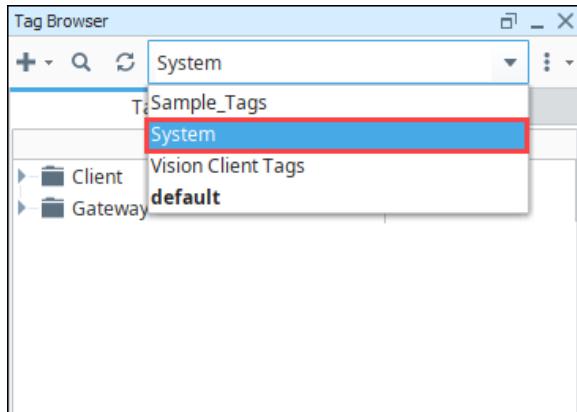
Related Topics ...

- [System Tags](#)
- [User Defined Types - UDTs](#)

In This Section ...

System Tags

System Tags provide status about the Ignition system, such as memory usage, performance metrics, and so on. System Tags cannot be deleted or modified. To view the System Tags in the Tag Browser, go to the Tag Provider Selector and select **System**.



On this page ...

- [System Client Tags \(Vision Only\)](#)
- [Gateway System Tags](#)



System Tags

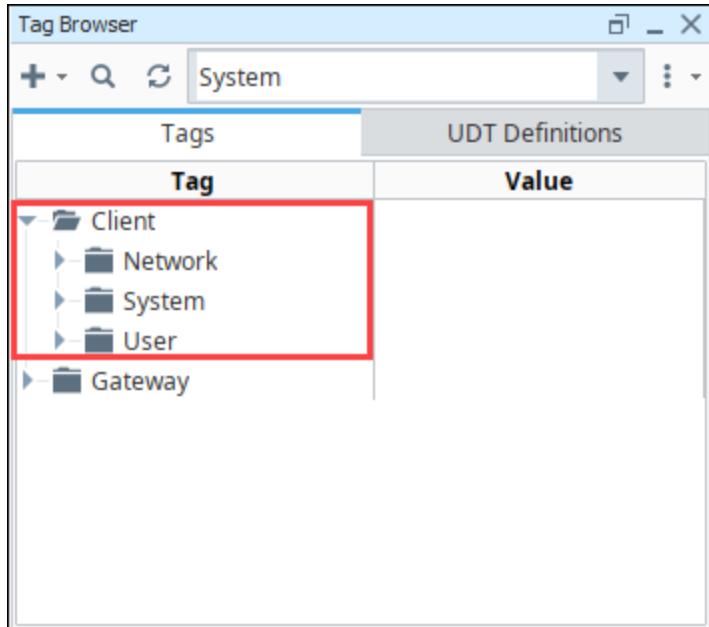
[Watch the Video](#)

The System Tags folders are displayed: **Client** and **Gateway**. The scope for each is slightly different.

Tags		UDT Definitions
Tag	Value	
Client		
Network		
System		
User		
Gateway		
Alarming		
Database		
Devices		
EAM		
Gateway Network		
OPC		
Performance		
Redundancy		
Sessions		
CurrentDateTime	2020-10-06 10:18...	
SystemName	Ignition-TR-89MC...	
Timezone	America/Los_Ange...	
UptimeSeconds	579,872	

System Client Tags (Vision Only)

Client-scoped System Tags provide status information about the client's system. They can be used with the Vision module for any Vision Client. Every individual client is going to have their own values, such as IP address, host name, username, and more. There are three folders within the System > Client folder: **Network**, **System**, and **User**. You cannot modify Client System Tags.



Vision System Client Tags

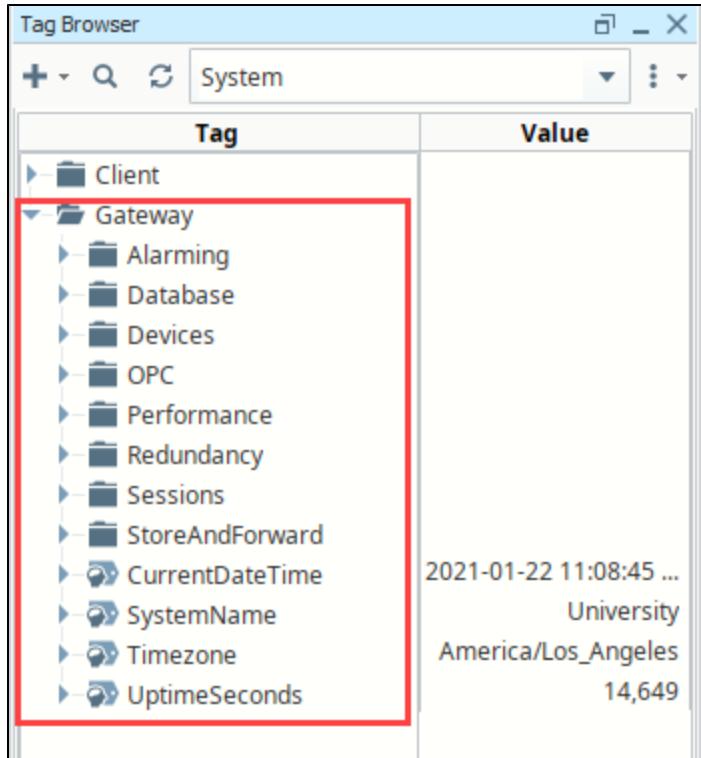
Tag	Description	Data Type
Network Folder		
GatewayAddress	Gateway URL address.	string
GatewayRedundancyRole	Redundancy State of the Gateway that the client is connected to. Independent, Master, Backup.	string
Hostname	Hostname (name) of the computer that the Client is running on.	string
IPAddress	IP Address of the computer that the Client is running on.	string
MACAddress	MAC Address of the computer that the Client is running on.	string
System Folder		
CurrentDateTime	Current system date and time. Format is yyyy-mm-dd hh:mm:ss a.	DateTime
DefaultDatabase	Name of the default database connection used by the project.	string
DefaultTagProvider	Name of the default Tag Provider used by the project.	string
FPMIVersion	Current Ignition version in use.	string
JavaVersion	Current Java version in use by the client.	string
LastProjectUpdate	Date of the last received update notification.	DateTime
OperatingSystem	Operating system of the computer that the Client is running on.	string
LastProjectUpdate	Date of the last received update notification.	DateTime
ProjectName	Name field for the current project.	string
ProjectTitle	Title field for the current project.	string
SystemFlags	A byte array of flags for the current state of the Client.	integer
UserSource	Name of the user source for the current Client.	string
User Folder		
Country	Two letter country code according to operating system. for example: US.	string
CurrentWindow	The current main window open in the project (top most Floating, Maximized window).	string
DateFormatFull	Full date format according to the operating system. Format: EEEE, MMMM d, y.	string

DateFormatLong	Long date format: MMMM d, y.	string
DateFormatMedium	Medium date format: MMM d, y.	string
DateFormatShort	Short date format: M/d/yy.	string
DateTimeFormatFull	Full date and time format: EEEE, MMMM d, y 'at' h:mm:ss a zzzz.	string
DateTimeFormatLong	Long date and time format: MMMM d, y 'at' h:mm:ss a.	string
DateTimeFormatMedium	Medium date and time format: MMM d, y, h:mm:ss a zzzz.	string
DateTimeFormatShort	Short date and time format: M/d/y, h:mm a.	string
HomeFolder	Home folder according to OS. For example: C:\Users\psmith.	string
Language	Language according to OS. For example: "en" for English.	string
OSUsername	OS user name, for example: PSmith.	string
RolesDataSet	Dataset with Roles for currently logged in user. For example: Dataset[2R x 1C].	dataset
RolesString	Comma separated string with Roles for currently logged in user. For example: Administrator, Operator.	string
TimeFormatFull	Full time format according to the operating system. Format: h:mm:ss a zzzz.	string
TimeFormatLong	Long time format: h:mm:ss a z.	string
TimeFormatMedium	Medium time format: h:mm:ss a.	string
TimeFormatShort	Sort time format: h:mm a.	string
Timezone	Current timezone, for example, America/Los Angeles.	string
Username	Currently logged in username, for example, PSmith.	string

Gateway System Tags

Gateway System Tags exist in the Gateway scope. There are nine folders within the Gateway Tags folder: **Alarming**, **Database**, **Devices**, **EAM**, **Gateway Network**, **OPC**, **Performance Redundancy** and **Sessions**.

The following Gateway-scoped System Tags are provided with Ignition. They can be used anywhere within Ignition.



Gateway System Tags

Tag	Description	Data Type
CurrentDateTime	Current system date and time. Format is yyyy-mm-dd hh:mm:ss a.	DateTime
SystemName	Computer name where the Gateway is installed.	String
Timezone	Timezone on the Gateway computer. For example, America/Los Angeles.	String
UptimeSeconds	Number of seconds since Ignition was started.	Long
Alarming		
Active and Acked	Number of alarms currently active and acknowledged.	Integer
Active and Unacked	Number of alarms currently active and unacknowledged.	Integer
Clear and Acked	Number of alarms cleared and acknowledged.	Integer
Clear and Unacked	Number of alarms cleared and unacknowledged.	Integer
Database		
<p>Note: There will be a subfolder for each database connection, or none if there are no connections. Each subfolder will have the following Tags.</p>		
ActiveConnections	Number of active connections in the pool to this database connection.	Integer
Available	Indicates whether this datasource is available.	Boolean
AvailableThroughFailover	Indicates if any database along the failover chain attached to this data source can be reached.	Boolean
AvgQueryTime	Average time, in seconds, that it is taking database queries to run.	Integer
ConnectionSaturation	Percentage of possible query throughput that is being used (ratio of currently active connections to maximum possible connections).	Double
QueriesPerSecond	Number of queries running per second.	Integer

Devices		
Note: For each device configuration on the gateway, a separate subfolder will exist. The contents of each subfolder are listed below.		
Description	Description of the device connection as configured on the device connection on the Gateway.	String
Enabled	Boolean representing whether or not the device connection is enabled.	Boolean
Name	Name of the device connection.	String
Status	Status of the connection.	String
Gateway Network		
Note: Each system connected over the gateway network receives a special folder, containing the following Tags:		
IsAvailable	Whether the remote gateway is available or not.	Boolean
LastComm	A timestamp representing the last known communication received from the remote Gateway.	DateTime
OPC		
Note: There will be a subfolder for each OPC UA Server. Each subfolder will have the following Tags.		
Connected	Whether the OPC UA server is connected to Ignition.	Boolean
Enabled	Whether the OPC UA server connection is enabled.	Boolean
State	The state name of the connection. For example: Connected, Faulted, Connecting.	String
Performance		
Available Disk Space (MB)	Available disk space on the computer Ignition is installed on, in megabytes.	Long
CPU Usage	CPU Utilization as reported to the Java Virtual Machine.	Double
Disk Utilization	Percentage of hard disk that is in use.	Double
Max Memory	Maximum amount of RAM the Gateway can use, in megabytes.	Long
Memory Usage	Amount of RAM currently in use by the Gateway, in megabytes.	Long
Memory Utilization	Current memory usage/maximum memory usage.	Double
Redundancy		
Connection, Is Connected	Whether this Gateway is connected to another for redundancy.	Boolean
Connection, PeerId	The ID of the Gateway connected to, empty string if not connected.	String
ActivityLevel	Indicates where the Gateway is in the redundant state. Can be undecided, cold, warm, hot, or active.	String
IsActive	Whether the Gateway is running.	Boolean
IsMaster	Whether the Gateway is the master. False if the backup is in control.	Boolean
Role	Named role of the Gateway. Options: Independent, Master, Backup.	String
Sessions		
SessionCount	Number of active sessions on this Gateway. Designers, Clients, and Sessions.	Integer
Store and Forward		

This feature is new in Ignition version **8.1.2**
[Click here](#) to check out the other new features

The Store and Forward System Tags were added in version 8.1.2

Note: There will be a subfolder for each database connection, or none if there are no connections. Each subfolder will have the following Tags.

Database Storage	Database storage for this Store and Forward engine.			
	Tag	Description		Data Type
	Available	Indicates whether Database Storage is available for this Store and Forward engine.		Boolean
	isStore	Indicates if the data storage sink is able to store records. This data store is used for the optimization of the records before they are forwarded to the database and no data is technically stored in it. For this reason, this tag's value will always be false.		Boolean
Local Cache	Local cache for this Store and Forward engine.			
	Tag	Description		Data Type
	ForwardMetrics Folder	Tag	Description	Data Type
		AverageDuration	The average duration for records to be forwarded to the data sink from the local cache.	Double
		MaxDuration	The maximum duration for records to be forwarded to the data sink from the local cache.	Double
		MinDuration	The minimum duration for records to be forwarded to the data sink from the local cache.	Double
		Throughput	Number of records forwarded to the data sink per second. Throughput will be -1 if idle.	Double
	StorageMetrics Folder	TimeUnit	Unit of time for this variable. The unit of time for this variable is seconds.	String
		Total	Total number of records forwarded to the data sink from the local cache.	Long
		Tag	Description	Data Type
		AverageDuration	The average duration for records to be forwarded to the data sink from the local cache.	Double
		MaxDuration	The maximum duration for records to be forwarded to the data sink from the local cache.	Double
		MinDuration	The minimum duration for records to be forwarded to the data sink from the local cache.	Double
	TakeMetrics Folder	Throughput	Number of records forwarded to the data sink per second. Throughput will be -1 if idle.	Double
		TimeUnit	Unit of time for this variable. The unit of time for this variable is seconds.	String
		Total	Total number of records forwarded to the data sink from the local cache.	Long

		AverageDuration	The average duration to read a record from the local cache.	Double			
		MaxDuration	The maximum duration to read a record from the local cache.	Double			
		MinDuration	The minimum duration to read a record from the local cache.	Double			
		Throughput	The number of records read from the local cache per second. Throughput will be -1 if idle.	Double			
		TimeUnit	Unit of time for this variable. The unit of time for this variable is seconds.	String			
		Total	Total number of records read from the local cache.	Long			
		Available	Indicates if the local cache for this Store and Forward engine is available.	Boolean			
		CanQuarantine	Indicates if this local cache can quarantine records. If the local cache cannot quarantine a record, the record will be dropped and lost forever.	Boolean			
		DroppedRecords	Indicates the number of dropped records for this local cache. A record is considered dropped if it can not be added to one of the buffers, (i.e., when a buffer is full and the Store and Forward engine can no longer accept new records).	Integer			
		IsStore	Indicates if the local cache is able to store records.	Boolean			
		MaxRecords	Maximum number of records this local cache can accept.	Integer			
		PendingRecords	Number of pending records for this local cache.	Integer			
		QuarantinedRecords	Number of quarantined records in this local cache. Quarantined data is data that has erred-out multiple times during attempts to forward it, or data that could not be stored because of some configuration issues.	Integer			
Memory Buffer	Memory Buffer Folder Tags						
		Tag	Description			Data Type	
	ForwardMetrics Folder	Tag	Description				
		AverageDuration	The average duration for records to be forwarded to the data sink from the memory buffer.				
		MaxDuration	The maximum duration for records to be forwarded to the data sink from the memory buffer.				
		MinDuration	The minimum duration for records to be forwarded to the data sink from the memory buffer.				
		Throughput	Number of records forwarded to the data sink per second. Throughput will be -1 if idle.				
		TimeUnit	Unit of time for this variable. The unit of time for this variable is seconds.				
	StorageMetrics Folder	Tag	Description				
		AverageDuration	The average duration to store and record into the memory buffer.				

		MaxDuration	The maximum duration to store a record into the memory buffer.	Double				
		MinDuration	The minimum duration to store a record into the memory buffer.	Double				
		Throughput	The number of records that go through the memory buffer per second. Throughput will be -1 if idle.	Double				
		TimeUnit	Unit of time for this variable. The unit of time for this variable is seconds.	String				
		Total	Total number of records stored in the memory buffer.	Long				
	Take Metrics Folder	Tag	Description	Data Type				
		Average Duration	The average duration to read a record from the memory buffer.	Double				
		MaxDuration	The maximum duration to read a record from the memory buffer.	Double				
		MinDuration	The minimum duration to read a record from the memory buffer.	Double				
		Throughput	The number of records read from the memory buffer per second. Throughput will be -1 if idle.	Double				
		TimeUnit	Unit of time for this variable. The unit of time for this variable is seconds.	String				
		Total	Total number of records read from the memory buffer.	Long				
	Available	Indicates if the memory buffer for this Stored and Forward engine is available.			Boolean			
	CanQuarantine	Indicates if this memory buffer can quarantine records. The memory buffer cannot quarantine records so this will always be false.			Boolean			
	DroppedRecords	Indicates the number of dropped records for this memory buffer. A record is considered dropped if it can not be added to one of the buffers, (i.e., when a buffer is full and the <u>Store and Forward</u> engine can no longer accept new records).			Integer			
	IsStore	Indicates if the memory buffer is able to store records.			Boolean			
	MaxRecords	Maximum number of records this memory buffer can accept.			Integer			
	PendingRecords	Number of pending records for this memory buffer.			Integer			
	QuarantinedRecords	Number of quarantined records for this memory buffer. Quarantined data is data that has erred-out multiple times during attempts to forward it, or data that could not be stored because of some configuration issues.			Integer			
Available	Indicates if this database engine is available.				Boolean			
Dropped	Number of quarantined records for this Store and Forward engine.				Integer			
Quarantine	Number of quarantined records for this Store and Forward engine.				Integer			

Related Topics ...

- [Vision Client Tags](#)
- [Types of Tags](#)

Creating Tags

Tags are created in the Designer in one of two ways; either using the Tag Creator or creating Tags manually in the Tag Browser.

The Tag Creator allows you to browse your connected devices and drag OPC Tags into your Tag Provider, as well as creating folders and other Tags manually. You can also make immediate changes to the Tag properties in the Tag Creator.

You can also create Tags manually in the Tag Browser. The Tag Browser allows you to choose from a list of these standard Tag types and change the Tag properties directly in the Tag Editor.

This page describes both methods for creating Tags.

On this page ...

- [Creating Tags in the Tag Creator](#)
- [Creating Tags Manually](#)
- [Editing Tags](#)
- [Addressing Bits](#)



INDUCTIVE
UNIVERSIT

Creating OPC Tags

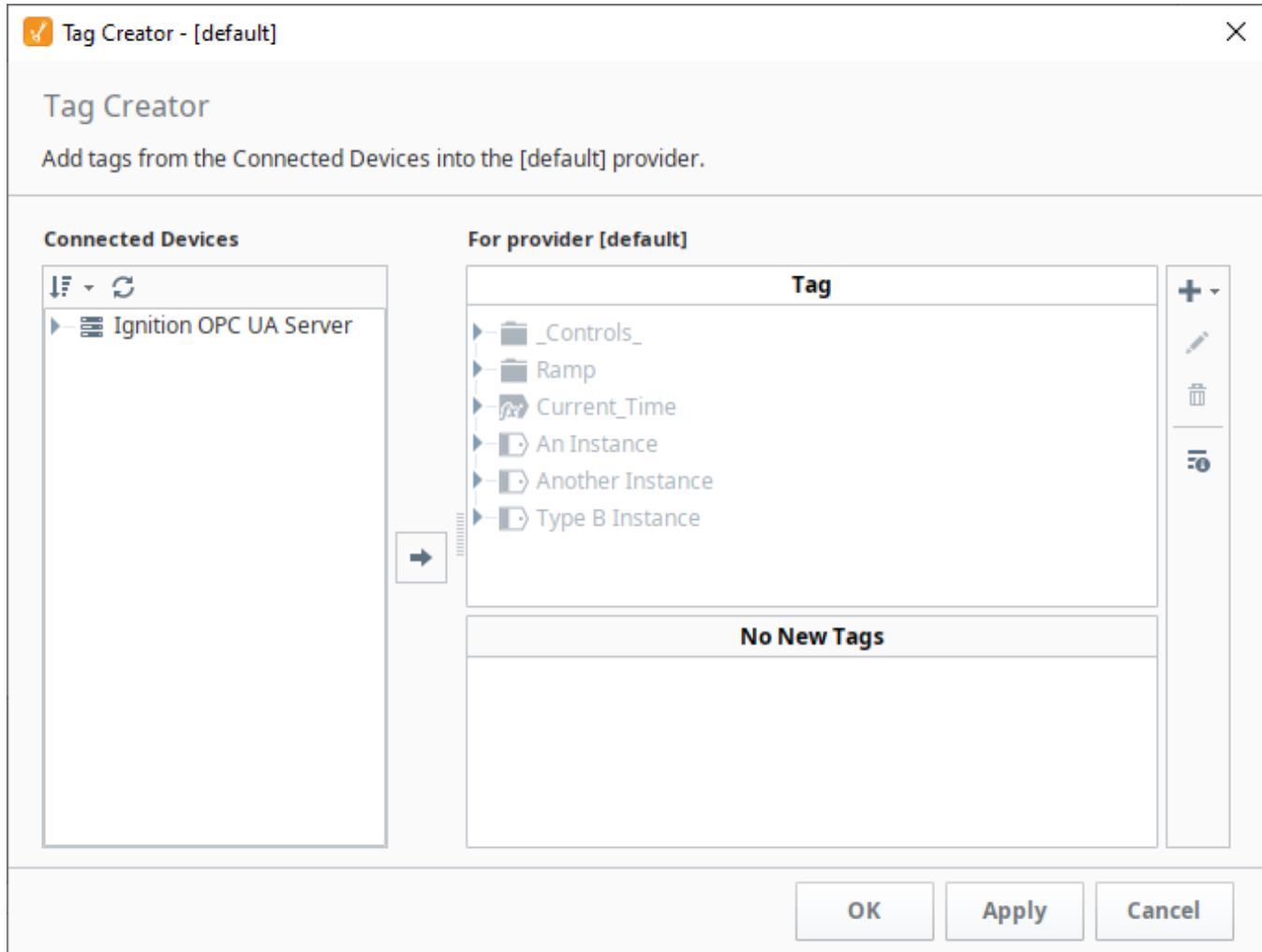
[Watch the Video](#)

Creating Tags in the Tag Creator

This feature is new in Ignition version **8.1.0**

[Click here](#) to check out the other new features

The Tag Creator lets you browse all your connected devices and OPC Servers for datapoints. When the Tag you're interested in is not available for browsing, you can manually add a Tag and change any Tag properties directly in the Tag Creator interface. This approach is common for devices like Modbus and Siemens where memory addresses aren't browsable. This interface allows you to create any standard type of Tag. Notably, Vision Client Tags do not use this Tag Creator.



Tags that are grayed out in the **For Provider** area are Tags that already exist in your Tag Browser. The Tag Provider represented here is dependent on the Provider selected by the Tag Browser before the Tag Creator was opened.

New Tags added by the Tag Creator will appear in black font. These new Tags are effectively in a "staging" area, meaning they won't be added to the Tag Provider until you press **OK** or **Apply**. In the Tag Creator window, you can also select the Preview icon to see a summary of new Tags you are adding.

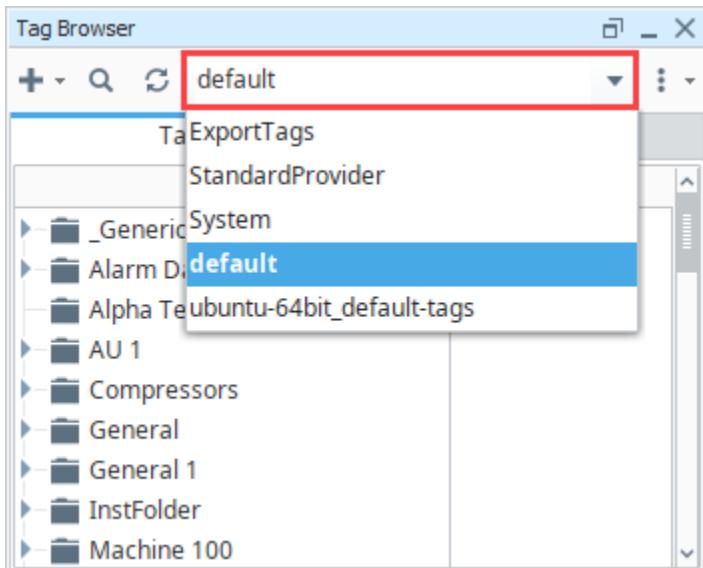
You have the option of editing your new Tags in the Tag Creator by selecting the Tag you want to edit in the **For Provider** area and double clicking or clicking on the edit (pencil) icon on the right. The Tag Creator edit window will open. If you have multiple Tags selected, only the first will be edited. Here you can edit any properties of the Tag. When you're finished, click **Commit** to accept your changes. When you are done adding Tags, click the **OK** or **Apply** buttons in the lower right corner to add your Tags to the Tag Browser.

The example below describes creating Tags using the Tag Creator.

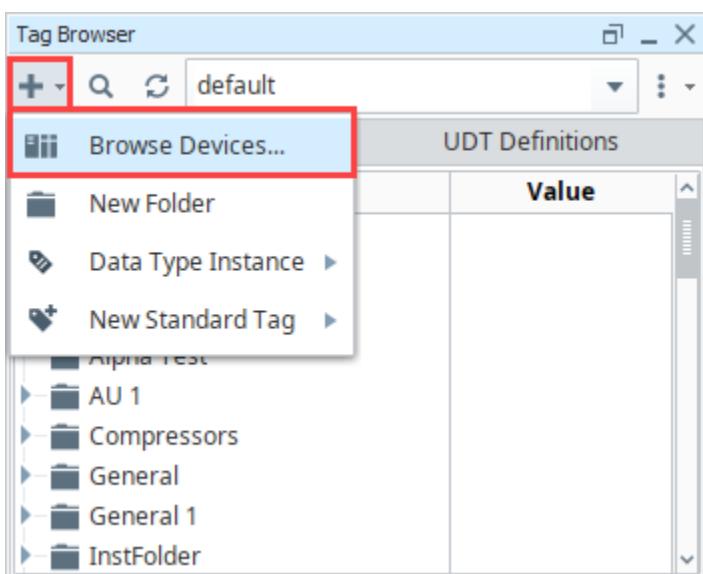
Note: In order to have any items appear under the **Connected Devices** tree, you must have a device connection.

1. In the Tag Browser, choose a Tag Provider from the dropdown list where you want to place your new Tags. The project's default provider will appear in **bold**.

Note: Tag Creator is not available when the 'System' or 'Vision Client Tags' providers are selected.

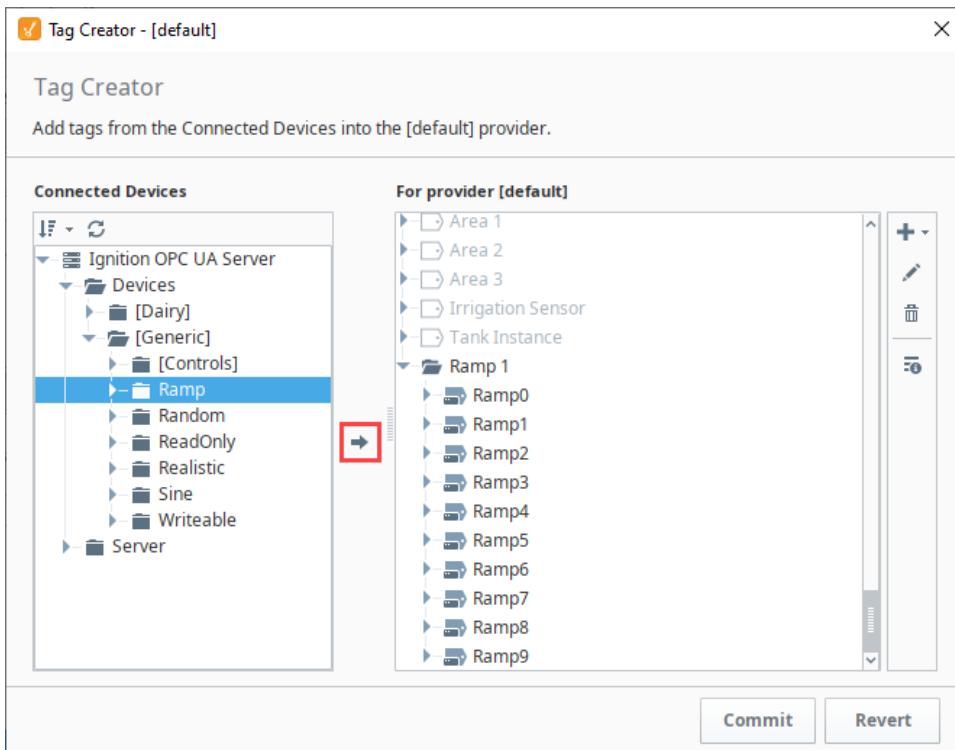


2. In the Tag Browser, click the Add icon to open the context menu. Here you can Browse Devices to add OPC Tags, or simply create a Tag or folder. Select **Browse Devices...**

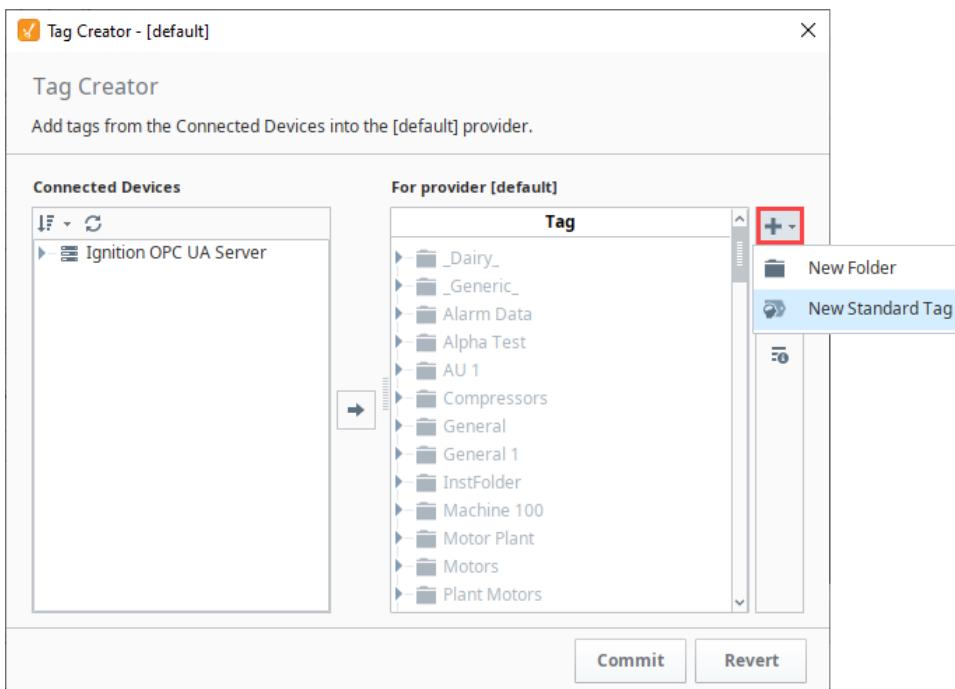


3. The Tag Creator window will open. There are two options described: Adding a OPC Tag and creating a standard Tag.

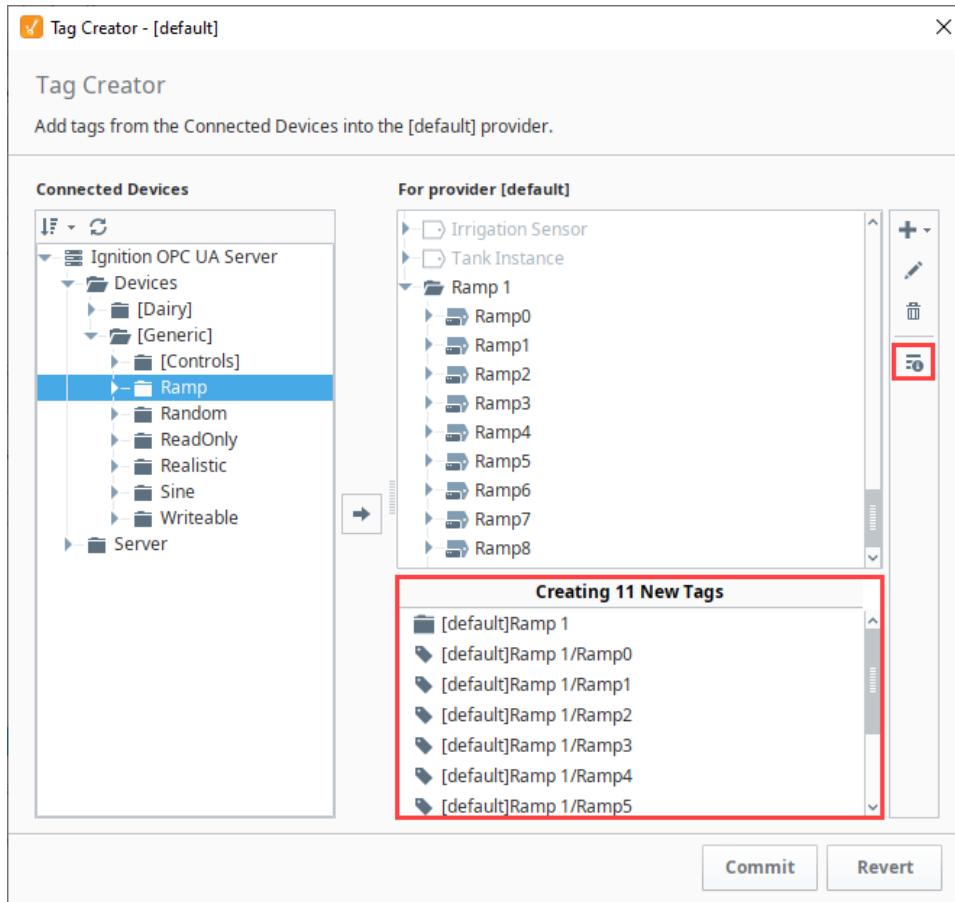
- This is where you can browse OPC Tags on your connected devices. Browse the devices and find some Tags that you're interested in. Select the folder(s) or Tag(s) you want to add. You can move the selected OPC Tags into your selected **For provider** area by clicking the **Right Arrow** icon or dragging them into the right panel. To unselect a row, click out on a blank area, thus clearing any selected row.



- b. You also have the option of creating a New Standard Tag in the Tag Creator by clicking the **Add** icon. A new blank Tag will appear in the **For provider** area in the currently selected folder (if one is selected), or at the bottom of the Tag list. See step 5 for editing Tags.



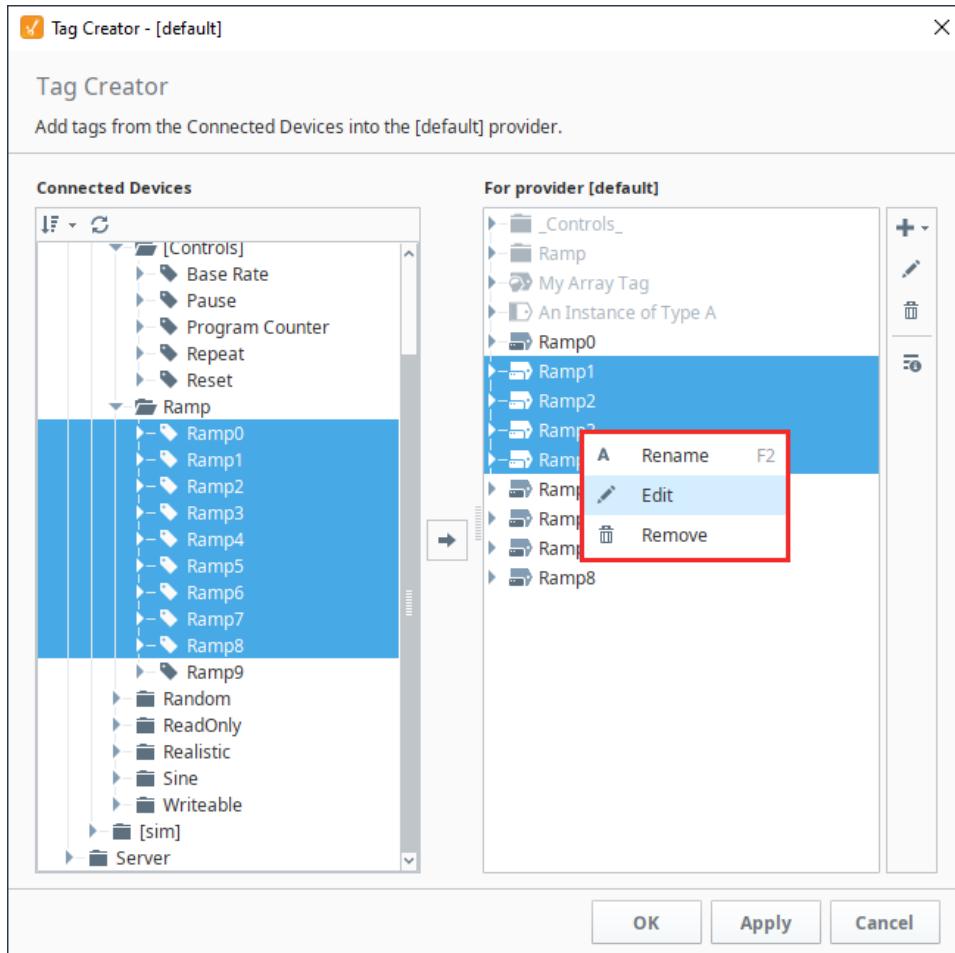
4. To Preview the Tags you want to add to your Tag Browser, scroll down to the bottom of the list in the For Provider area, or click the **Preview** icon on the right side of the window.



5. You have the choice of editing your Tags in the Tag Creator or editing them in your Tag Browser. To edit Tags in the Tag Creator, select a Tag and click on the **Edit** icon. Here you can change the Data Type, Item Path, set alarms or store history, and anything else that you can do to modify a Tag. Click the **Commit** button to submit your changes.

This feature is new in Ignition version 8.1.2
[Click here](#) to check out the other new features

As of 8.1.2, the **For Provider** area now features a right-click menu that also has **Edit** and **Remove** options.



- When finished, click the **Apply** button to add your new Tags to the Tag Browser. As soon as the Tags are in the Tag Browser, the values will start updating automatically.

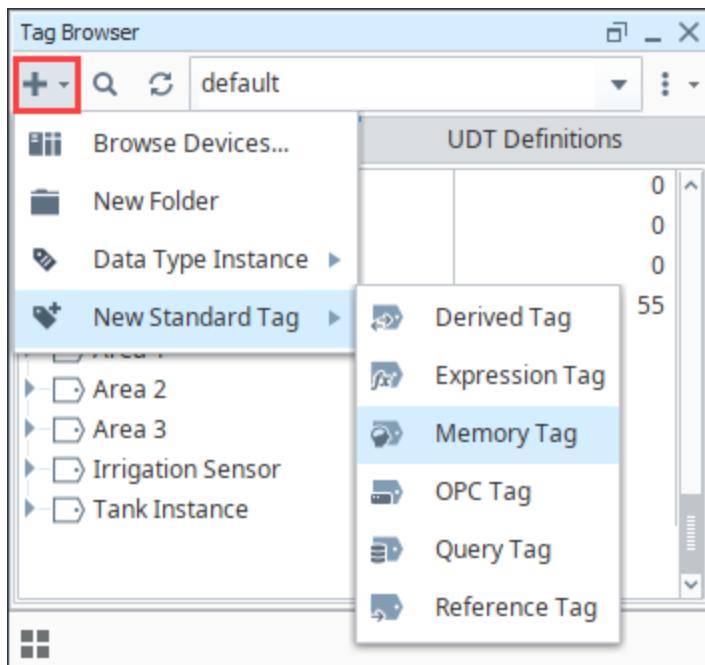
Tags		UDT Definitions
Tag	Value	
Dairy		
Generic		
Ramp		
Ramp0	9.26	
Ramp1	62.76	
Ramp2	1.85	
Ramp3	-3.83	
Ramp4	430.86	
Ramp5	452.41	
Ramp6	253.63	
Ramp7	387.78	
Ramp8	24.91	
Ramp9	1,615.74	
Ramp10	9.26	
Ramp11	9.26	

Creating Tags Manually

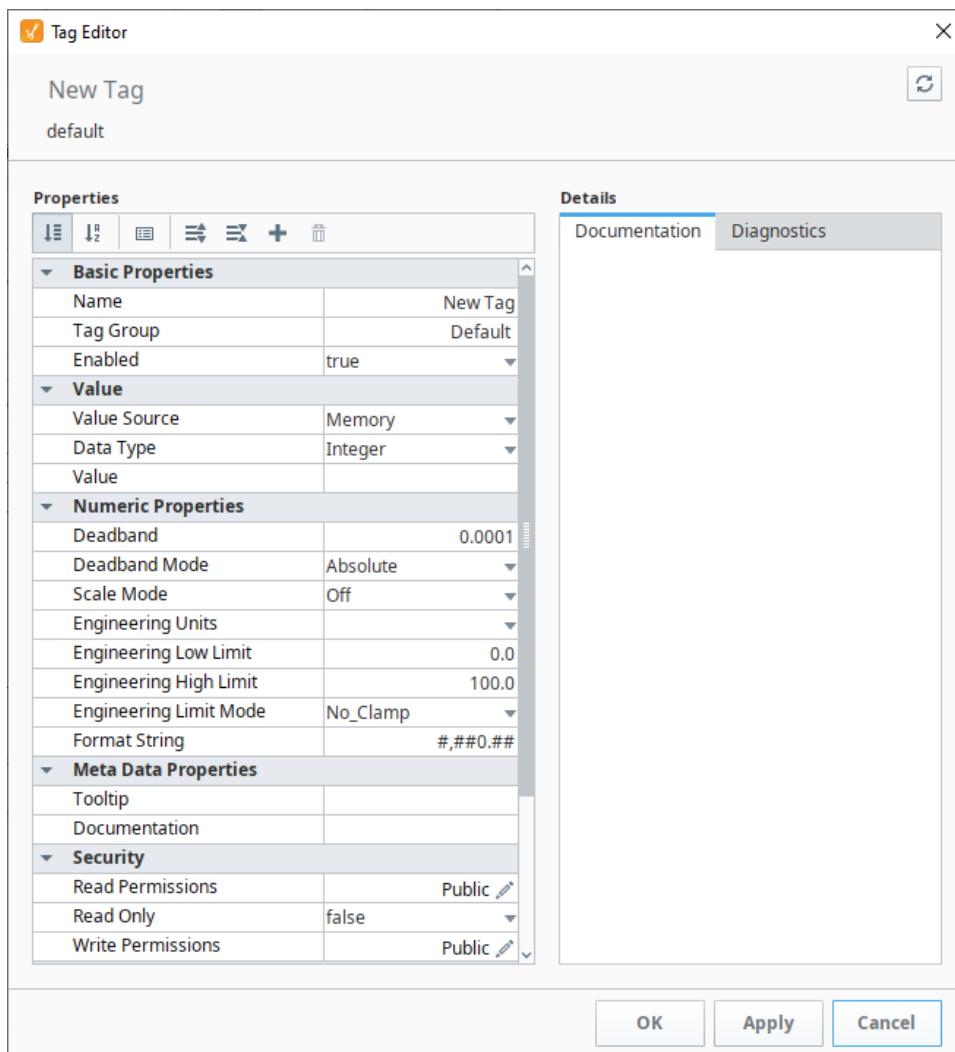
Tags can be created manually in the Tag Browser. You can create any of the Standard [Tag Types](#) or a Data Type Instance.

This example creates a Memory Tag.

1. You can create Tags in two different ways: click on the Add  icon to open the context menu and select New Standard Tag and choose the Tag type, or right click anywhere in the Tag Browser and select New Tag and choose the Tag type.

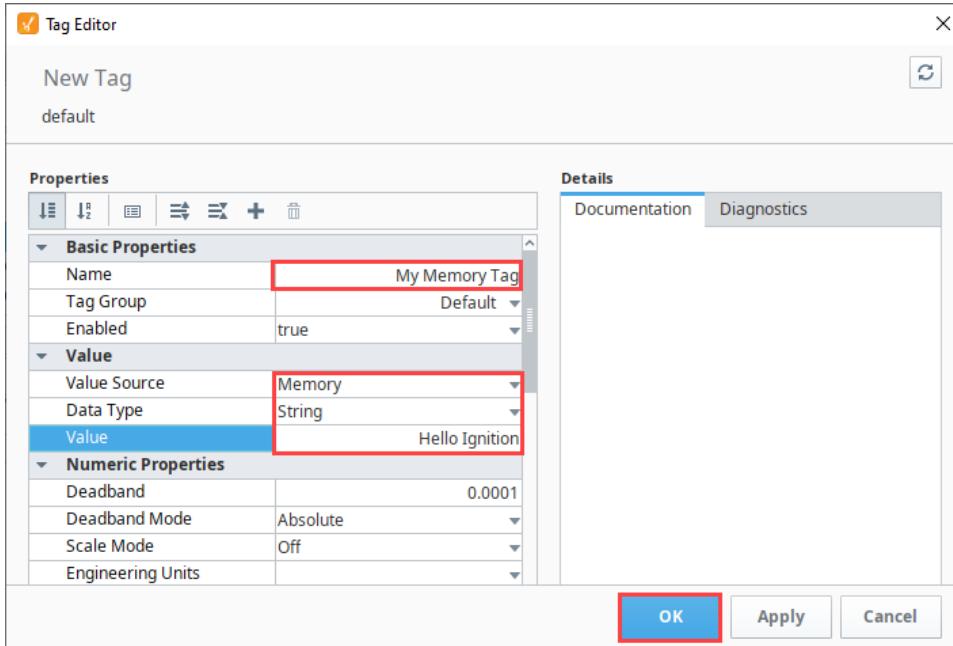


2. The Tag Editor will open to edit the Tag's properties.



3. Set the properties as follows:

Name: **My Memory Tag**
 Value Source: **Memory**
 Data Type: **String**
 Value: **Hello Ignition**



4. When finished, click **OK**. Your new Tag will be placed in the Tag column of the Tag Browser. Since this is a Memory Tag, its value will not change unless you write to it.

Editing Tags

The Tag Editor is a powerful tool used when creating Tags and for editing them. The properties displayed in the Tag Editor are custom to the type of Tag you've selected. You can find additional information on Understanding Tags and the Tag right click menu [here](#).

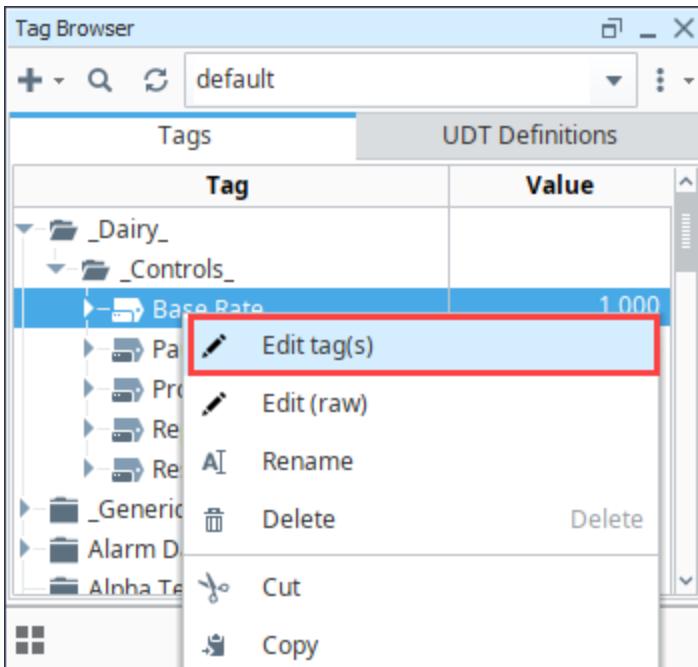
 INDUCTIVE
UNIVERSITY

The Tag Editor

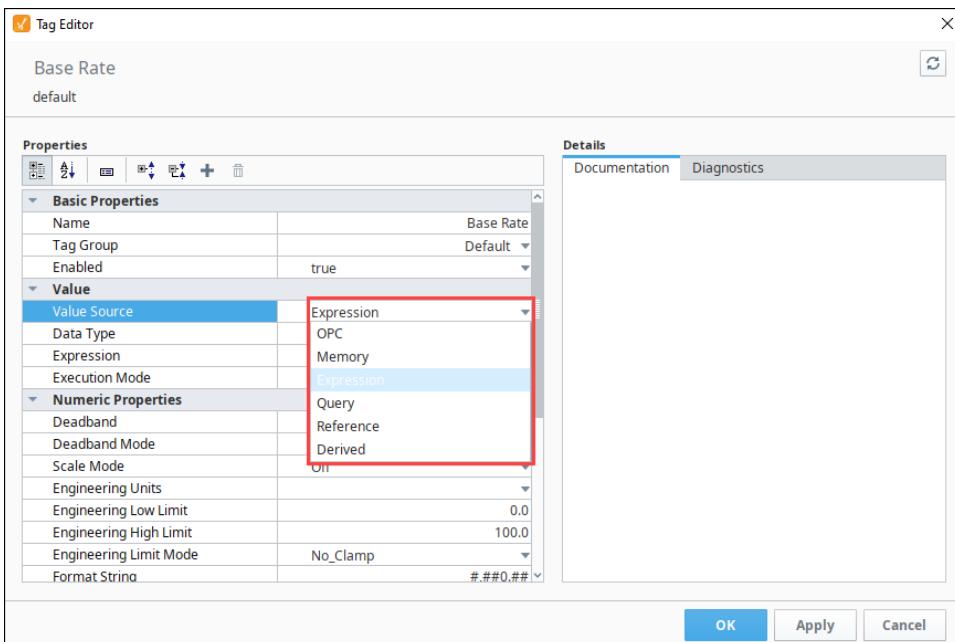
[Watch the Video](#)

Edit a Tag

- To edit an existing Tag, right-click on the Tag, and select the **Edit Tag** icon .



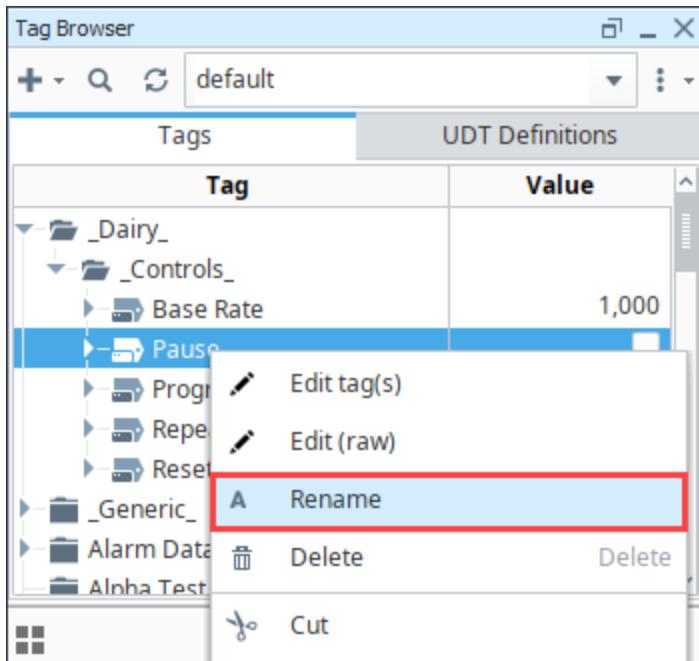
- Once in the Tag Editor, you can update the Tag properties. For example, if you want to change the Tag to a different type – such as from OPC to Expression – go to the Value Source property, click the Expand ▾ icon, and choose the type of Tag (OPC, Memory, Expression, Query, Reference, or Derived) that you want.



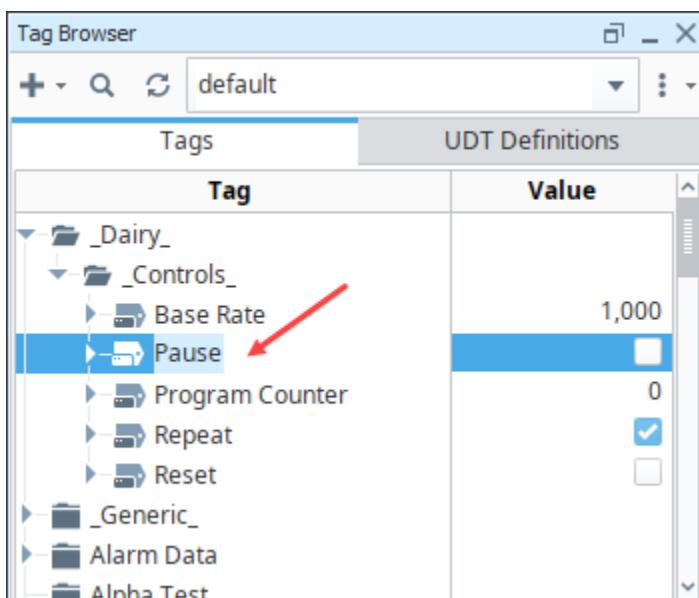
Rename a Tag

Tags names are flexible. For naming conventions, see Tag Browser.

- To rename an existing Tag, right click on the Tag in the Tag Browser and select the **Rename** option.



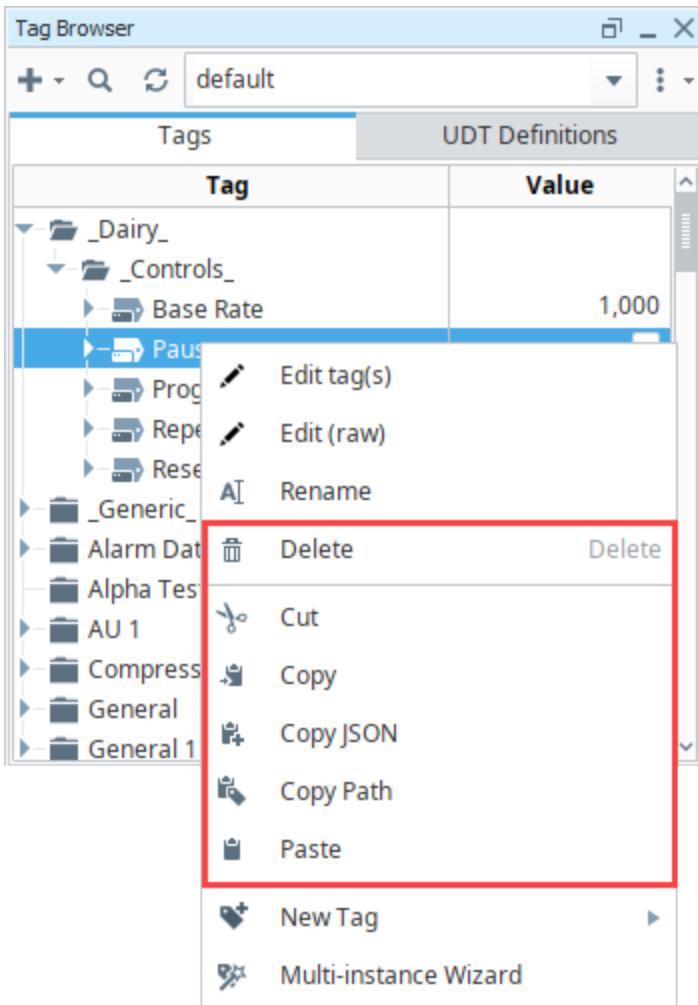
2. The cursor will now blink inside the Tag name and you can type the new name.



Cut, Paste, or Copy a Tag

You can also cut, paste, and copy Tags within the Tag Browser. Right click on the Tag in the Tag Browser. Choose the command you want.

- **Delete:** Completely removes the Tag.
- **Cut:** Delete the Tag from the current location, but leave it in the clipboard to be pasted elsewhere in the browser.
- **Copy:** Make a copy of the Tag and leaves it in the clipboard to be pasted elsewhere in the browser.
- **Paste:** Pastes the Tag you've cut or copied into the currently selected location in the Tag Browser.



Addressing Bits

In bindings and scripting there are ways to read a single bit of a word, but you can also have a Tag to read and write to a single bit. In order to address individual bits in Ignition, you must create a separate OPC Tag pointing directly to the specific bit in the PLC.

When the integer values that come from the OPC Tags are a series of binary bits, it is then possible to address each bit. For example, an integer value can have a 16-bit binary representation as shown here:

Integer	Bit level representation	How it works
4096	0,0,0,0,0,0,0,0,0,0,0,1,0,0,0	$2^{12} = 4096$
1025	1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0	$2^0 + 2^{10} = 1025$

Note:

Addressing bits may work differently depending on the type of device you are addressing. Most commonly you will either use /<bit> like /0 or /1, or [<bit>] like [0] or [1], or will use a . like .0 or .1. IE:

- Micrologix: [device]N7:1/0
- ControlLogix: [device]Folder/Tag.0
- Siemens: [device]I0.0



Addressing Bits

[Watch the Video](#)

Address an Individual Bit in a Micrologix

In this example, we want to address bits from a known integer value: 1025. This is represented as above, with the first (0) and eleventh (10) bits true and all others false. Our Gateway is connected to a MicroLogix PLC named MLX. To address an individual bit, do the following:

1. From the **Tag Browser** window, create an OPC Tag to have a specific value:

Data Type: **Integer**

OPC Item Path: **[MLX]B3:0**

2. Once created, set the Tag value to **1025**

3. Then create another new OPC Tag to show only the value of the first bit of our previous Tag:

Data Type: **Boolean**

OPC Item Path: **[MLX]B3:0/0** (for Micrologic, you can specify the bit as: 0 /0 or 0.0. That is, with a slash/ or a period.)

[MLX]B3:0/0 has a value of "1" or a Boolean value of "True" because the first bit is true (integer value is odd).

4. You can create a Tag for any of the other individual bits. For example, create a new OPC Tag with a Boolean value to the second bit of the original Tag as follows:

Data Type: **Boolean**

OPC Item Path: **[MLX]B3:0/1**

[MLX]B3:0/1 has a value of "0" or a Boolean value of "False" because the second bit is false.

Once you have your Tags created, try changing the boolean values and see the Integer Tag change.

User Defined Types - UDTs

What Is a UDT?

UDTs (User Defined Types), also referred to as Complex Tags, offer the ability to leverage object-oriented data design principles in Ignition. UDTs are extremely important in Ignition. With UDTs, you can dramatically reduce the amount of work necessary to create robust systems by essentially creating parameterized "data templates".

By defining UDTs and using these "data templates", you can generate Tag instances to rapidly build complex screens. A change to the type definition is then inherited by *all instances*, drastically saving time when making routine changes.

The UDT data types are fully supported by [Vision Templates](#), which means you can configure templates for your custom data types and take advantage of drag-and-drop binding to rapidly build complex screens.

Primary UDT Features

- **Object Oriented** - Use small or large groups of Tags to create a single object. Create objects that match your real world devices or the existing structures in your PLCs.
- **Central Definition** - Once you define your data type, you can then create instances of it. If at a later time you want to change some aspect of the type, you can simply edit the type definition, and all instances of it are automatically updated.
- **Parameterized Settings** - Define custom parameters on your data type, and then reference them inside some or all of your member Tags. When it comes time to create instances, you can simply modify their parameter values in order to change where the underlying data comes from.
- **Extendable** - Data types can inherit from other data types in order to add additional members or override settings. Instances can also override settings, allowing for flexibility when dealing with irregularities and corner cases.

On this page ...

- [What Is a UDT?](#)
- [Primary UDT Features](#)
- [UDT Terminology](#)
- [Creating a UDT Definition and Instance](#)
 - [Creating a Definition](#)
 - [Creating an Instance](#)
 - [Override Instance Properties](#)
 - [Make Changes to the Definition](#)
- [UDT Root Node Properties](#)
- [Assigning Colors to UDTs](#)
- [Binding to UDTs](#)



INDUCTIVE
UNIVERSITY

Understanding UDTs

[Watch the Video](#)

UDT Terminology

Many terms are frequently used when discussing UDTs:

Definition

A Definition represents the structure of a UDT. Definitions don't run, so Tags inside of a Definition won't poll or subscribe to anything. Rather they represent a Tag structure which Instances will inherit from. Changes made to a Definition are automatically applied to any Instances of that Definition.

In the Tag Browser, UDT Definitions are always located under the **UDT Definitions** tab.

Tag Browser	
+ - ⌂ ⌂ ⌂ default	
Tags	
Definition	Config
My Definition	
A Folder	
A Memory Tag	123
Some other Definition	
New Tag	

Instances

Instances are running copies of a Definition. All Instances have a "Parent Type", which is the definition that the Instance is inheriting from. The structure of an Instance is defined by its parent Definition, so you can not add new Members to an Instance. However, you can override the values on properties in any Member.

In the Tag Browser, UDT Instances can be found under the Tags tab, and are signified by either a plain white Tag icon, or a Tag icon with a vertical stripe. Furthermore, you can expand the UDT Instance and find the members (other Tags) in the UDT.

Tag Browser	
+ - ⌂ ⌂ ⌂ default	
Tags	
Tag	Value
A Standard Tag	
Instance With Color	123
Instance Without Color Defined	

Parameters

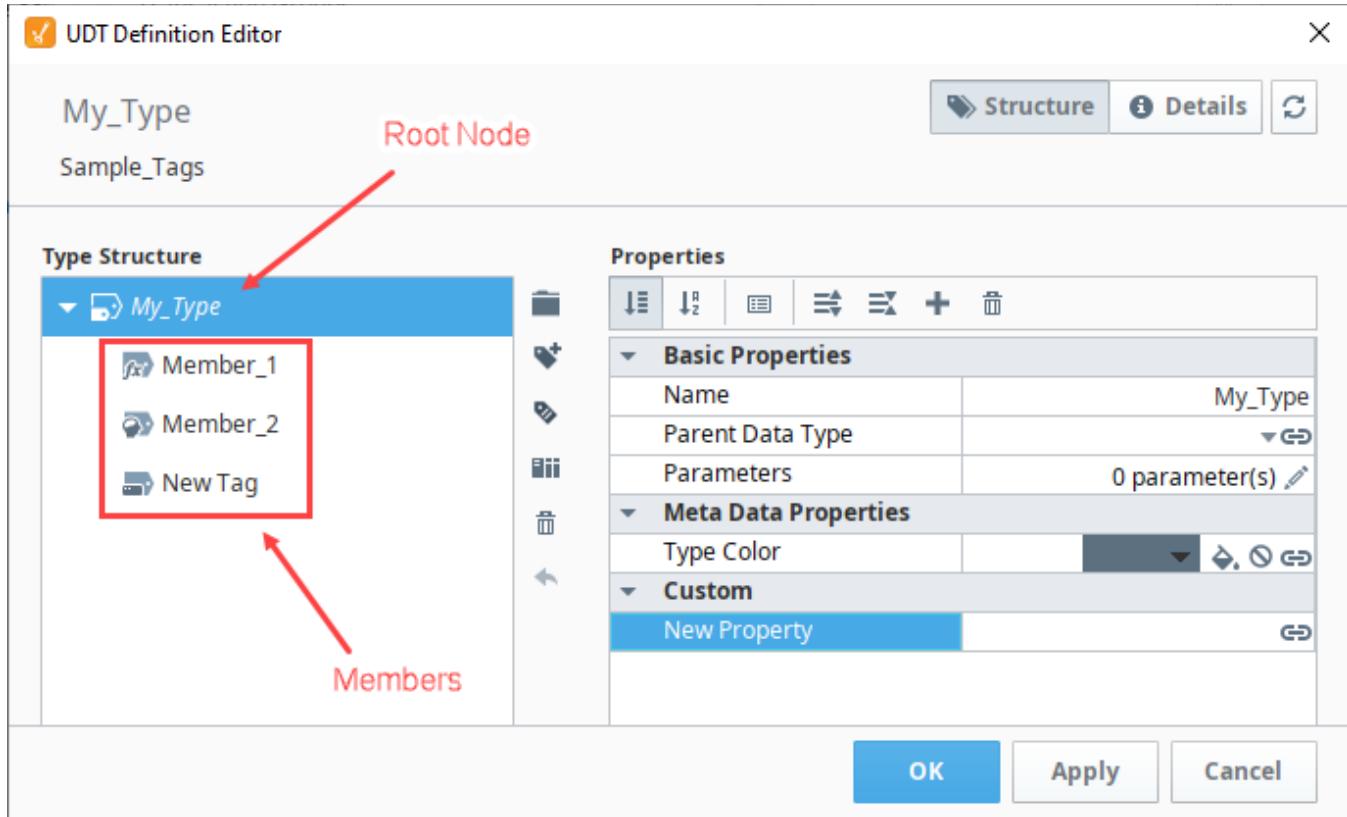
Parameters are user created properties that can be used to create parameterized data templates. Parameters are configured on Definitions, and their values can be overridden on individual Instances. You can replace values on a member in a UDT with a reference to a parameter, allowing for example, if a data type consists of three OPC Tags that only differ by a number in the path, you can use a parameter for the "base address", allowing instances to be created with only one setting.

Root Node

The top level item in a UDT.

Members

Members are the Tags inside of a data type or instance. Members are always under a Root Node. Members can be standard Tag types or an instance of another UDT.



Override

Instances are copies of a Definition, but in some cases you may wish to change the value of a property on a particular member (Tag) in a UDT instance. This is called Overriding the property, allowing the property to have a value that deviates from the Definition.

Creating a UDT Definition and Instance

In these series of examples, we will create a very simple UDT Definition, make an instance of it, and make some additional configuration changes.

Creating a Definition

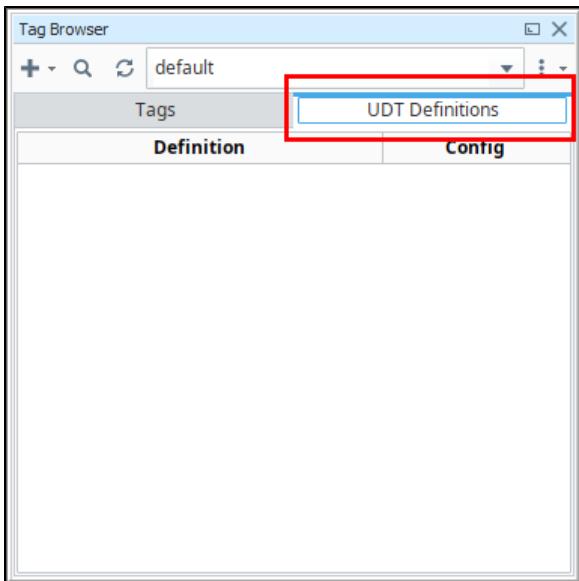
In this example, we'll demonstrate how to create a UDT Definition.

1. To create a UDT Definition, first switch click on the **UDT Definitions** tab in the Tag Browser. This is the only section where you can make UDT Definitions.

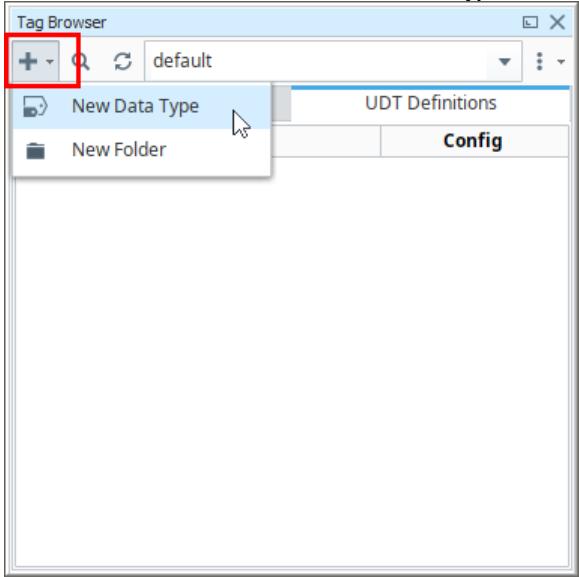

**INDUCTIVE
UNIVERSITY**

**Creating UDT
Definitions**

[Watch the Video](#)

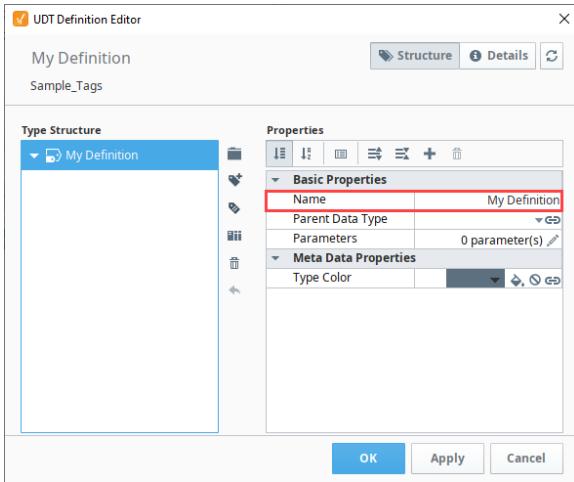


2. Next click the **Add** icon, and select **New Data Type** from the dropdown.

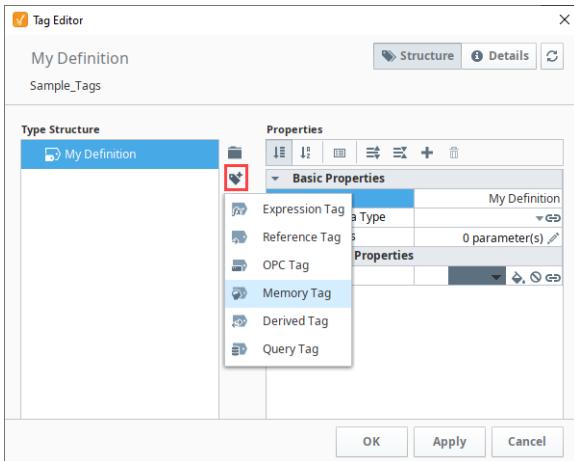


3. The Tag Editor will open, allowing you to create a new UDT Definition. To start, enter a name into the **Name** property. (For this example we used "My Definition".) This name is how the UDT will be identified by the rest of the system.

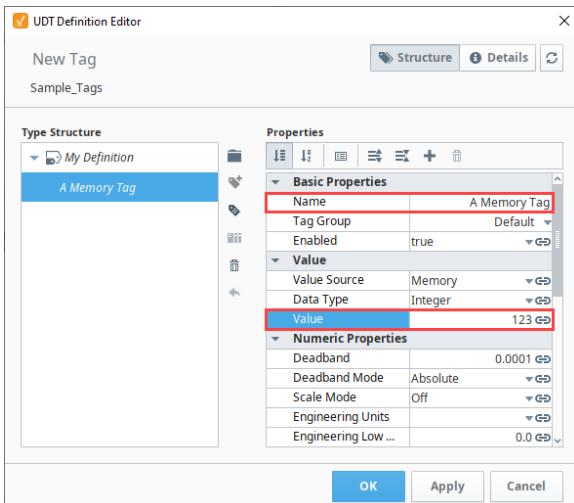
Note: Names can be changed later, but doing so after you've already made instances will create what's called an "orphaned UDT instance": an instance that is no longer associated with a definition. It's generally advised to avoid changing the name of a definition unless you're also willing to update any of the instances.



4. We'll keep this demonstration simple, and only add two members. We'll create a Memory Tag and add a Tag from a connected device. Let's create the Memory Tag first. Click the **Add** icon, and select **Memory Tag**.



5. The Tag Editor will now add a memory Tag to the **Type Structure** tree and select it, allowing the **Properties** table to show settings for the selected Tag.
 6. Change the Name to "A Memory Tag", and set a Value of 123.



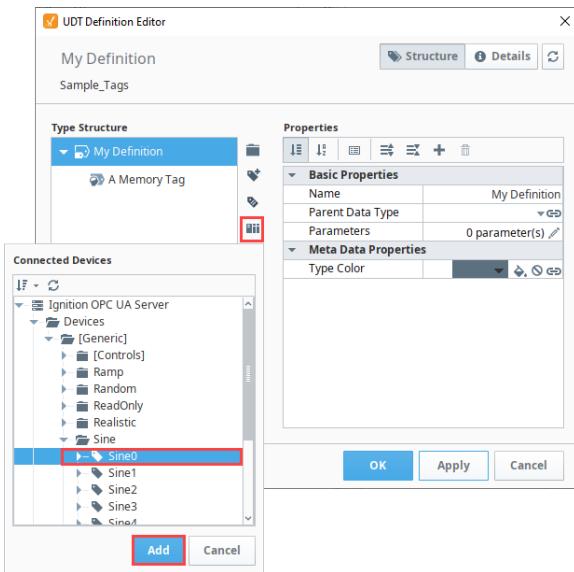
7. Now, let's add a Tag from the OPC UA server. Click the **Add** icon, and a window will open showing you your Connected Devices. Expand **Ignition OPC UA**

Server>Devices>Generic>Sine, and select **Sine0**. Click Add.

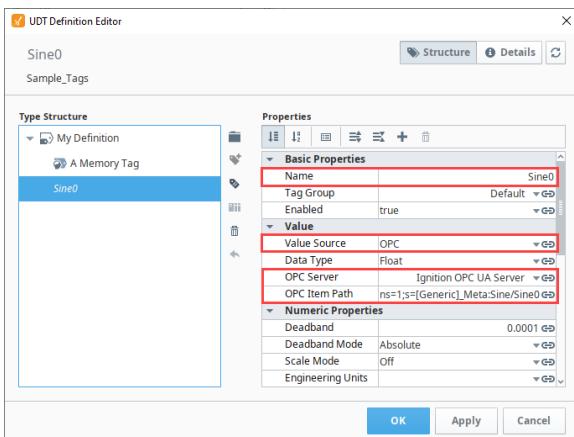
This feature is new in Ignition version **8.1.1**
[Click here](#) to check out the other new features

As of 8.1.1, you can browse OPC devices from within the Tag Editor in order to add OPC nodes to UDT definitions.

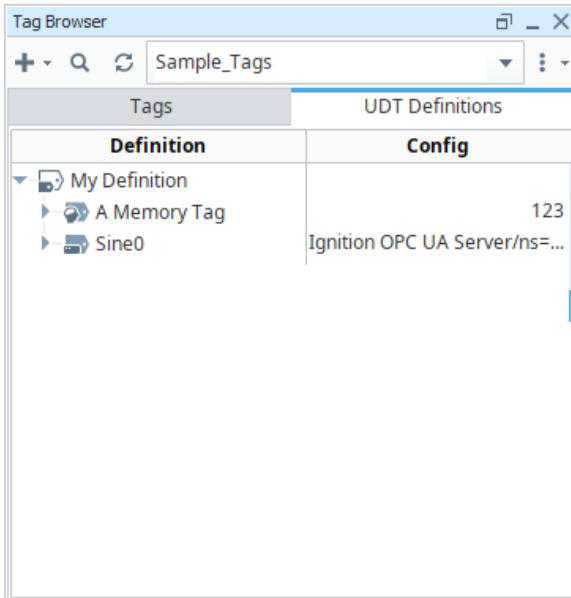
Note: When adding Tag(s) from a device, it requires that you have some connected devices such as the Ignition OPC UA Server as in this example.



8. You'll see that the **Sine0** Tag was added to the **Type Structure**. Select the **Sine0** Tag and you'll see all the property settings for that Tag.



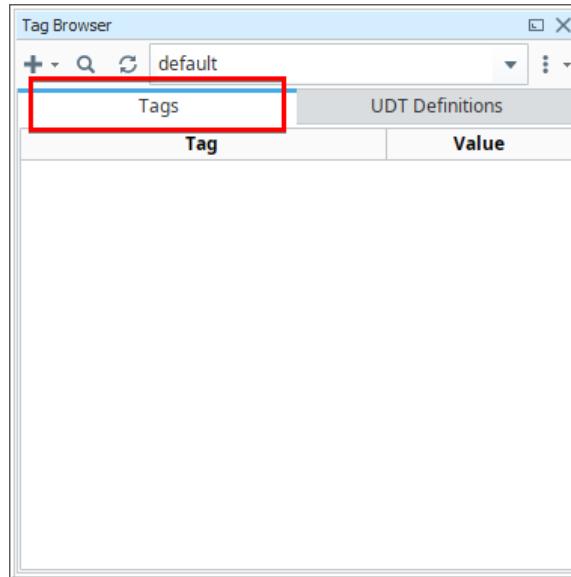
9. Click **OK**. This will close the Tag Editor, and apply your changes. The Tag Browser will now show your UDT Definition. You can expand the "My Definition" item to make the Memory and Sine0 Tags visible.



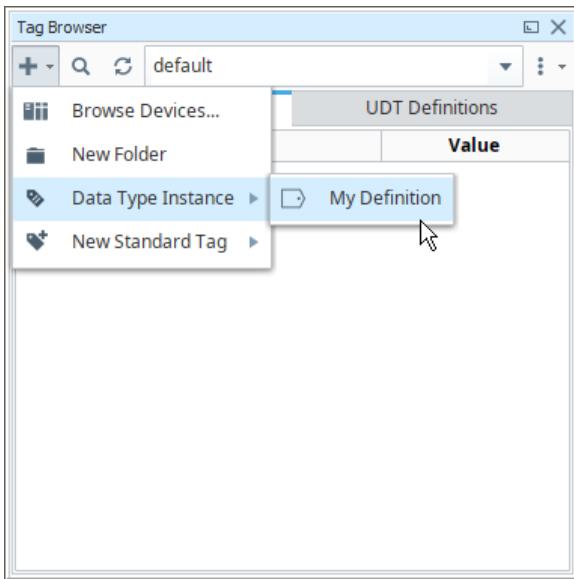
Creating an Instance

Now that we have a UDT Definition, we can create a UDT Instance. You can use the [Multi-Instance Wizard](#) to make many instances quickly, but you can also create a single instance.

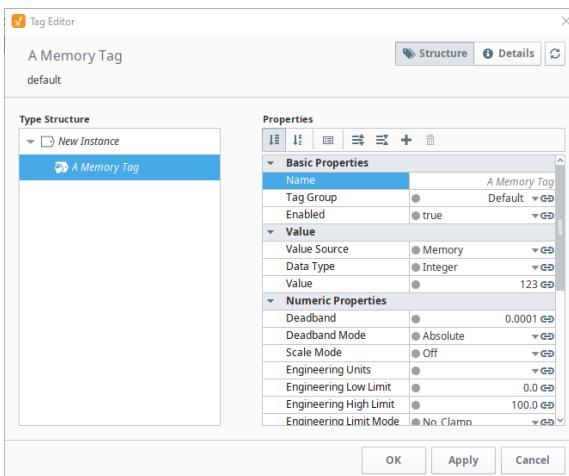
1. In the Tag Browser, switch back to the **Tags** tab. This section of the Tag Browser is where you create UDT instances. Definitions can not be placed in this section.



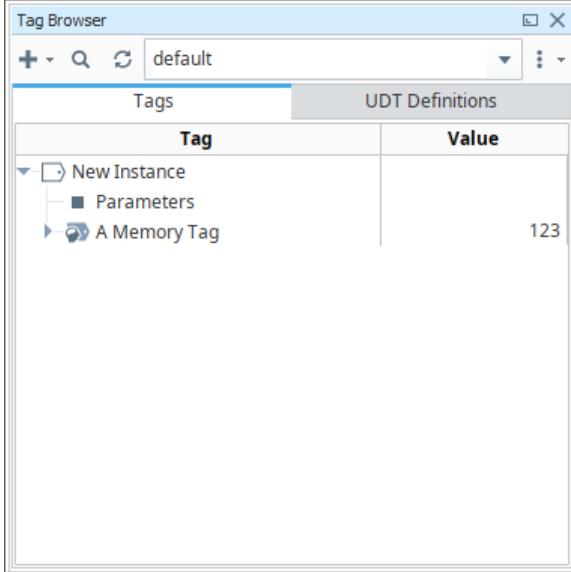
2. Click the Add icon and hover the mouse cursor over **Data Type Instance**. This will expand and show all of the UDT Definitions that exist within the active **Tag Provider**. Select the Definition we created in the previous example.



3. Again, the **Tag Editor** will open. This time allowing you to edit an instance of the UDT. From here you can name the Instance. In this case we'll leave it with the default name of "New Instance".
4. If you select the member ("A Memory Tag"), you'll notice that many of the properties have a grey dot. This signifies that the properties are inheriting their values from the definition.



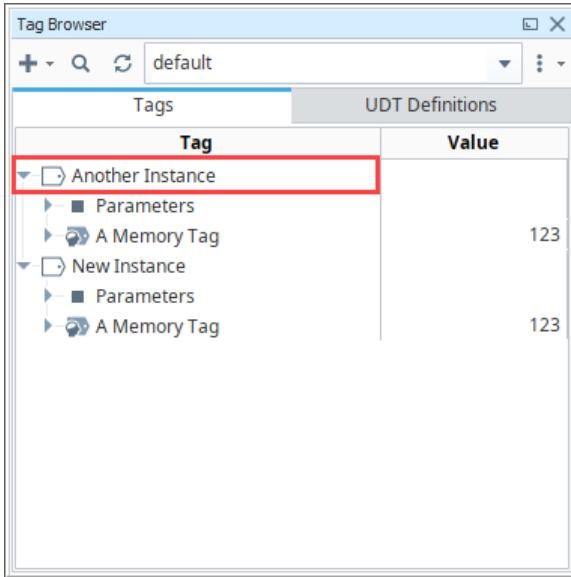
5. We won't make any changes here. Just click **OK**, which closes the Tag Editor and creates a UDT instance.



Override Instance Properties

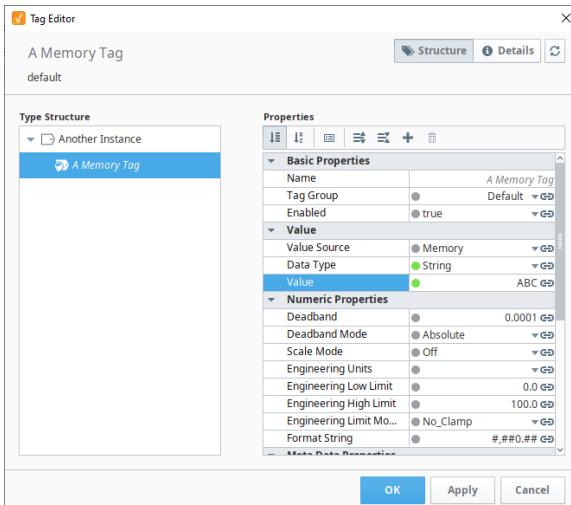
Instances inherit their structure and properties from the Definition, but property values can be overridden on an instance.

1. Now that we have an instance, let's create another. You can use the same steps in the last example, or just copy and paste **New Instance** from the Tag Browser's right-click menu.
2. Name the new instance "Another Instance".
3. If you need to make changes to a UDT Instance, you can open the **Tag Editor** by double clicking on the Instance, or any of its members.

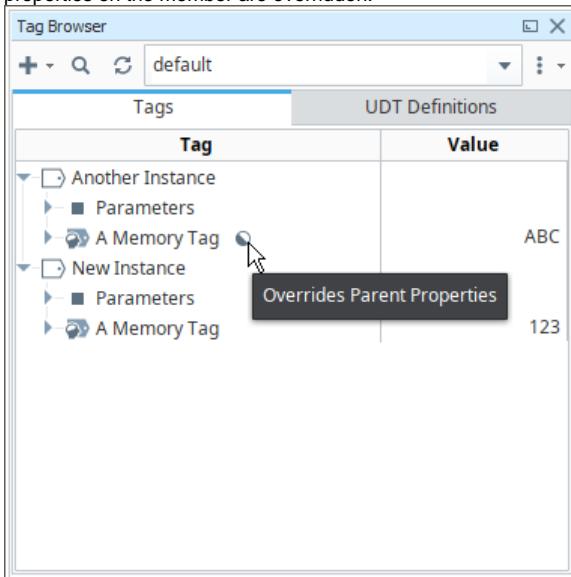


4. With both instances expanded, we can see that aside from the names, they're identical. Let's change that. Double click on **Another Instance** or any of its members to open the Tag Editor.
5. With the Tag Editor open and focused on **Another Instance**, select the memory Tag. Change the Data Type to a **String**, and the Value to "**ABC**".

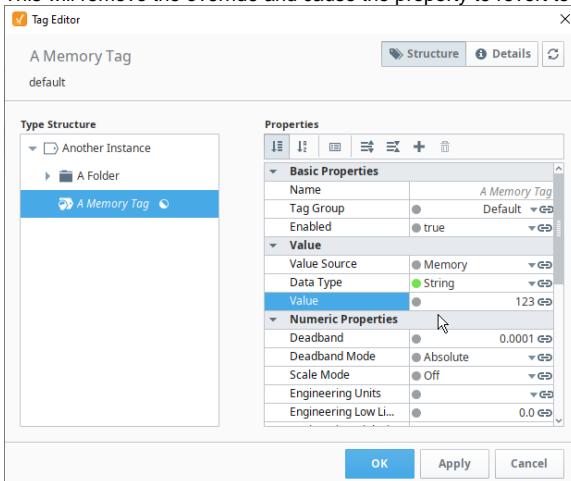
You'll notice that as you make changes to the properties, the grey dots become green signifying that the property values now differs from the Definition. This concept is known as "Overriding" a UDT member property.



6. Click **OK** to close the Tag Editor and apply your changes. You'll notice that the memory Tag in the instance we edited now has an attribute icon, signifying that one or more properties on the member are overridden.



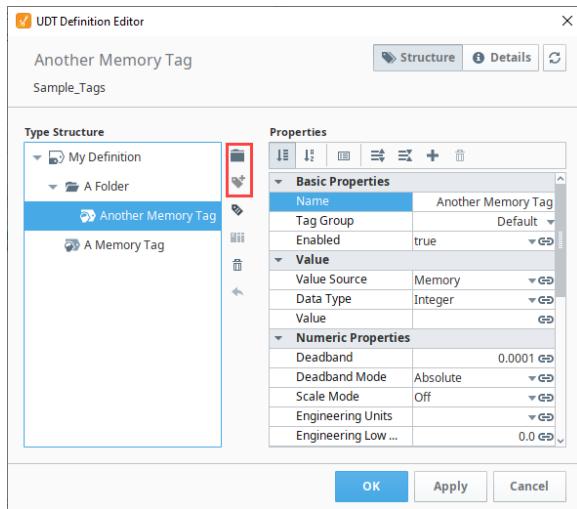
7. If you wish to remove the overrides, simply edit the Tag again, and click on the green dot. This will remove the override and cause the property to revert to the value on the Definition.



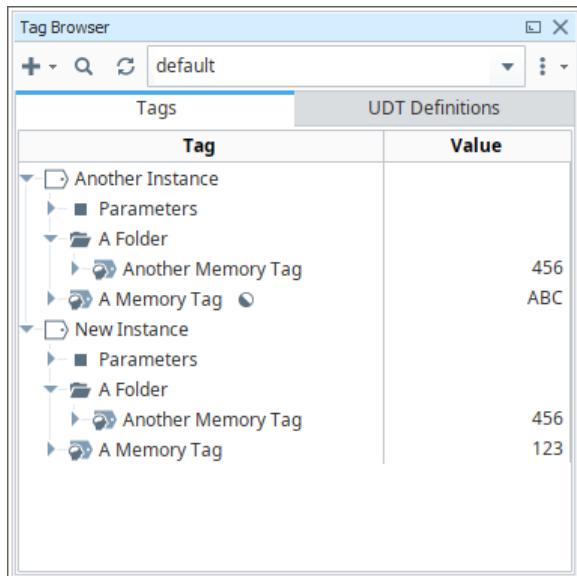
Make Changes to the Definition

Now that we have some instances, we can make a modification to the Definition, which will propagate the change down to the Instances.

1. Switch back to the **UDT Definition** tab in the Tag Browser.
2. Edit the **My Definition** UDT.
3. Use the **Add Folder** and **Add Standard Tag** icons to add a new folder, and a member inside of that folder.

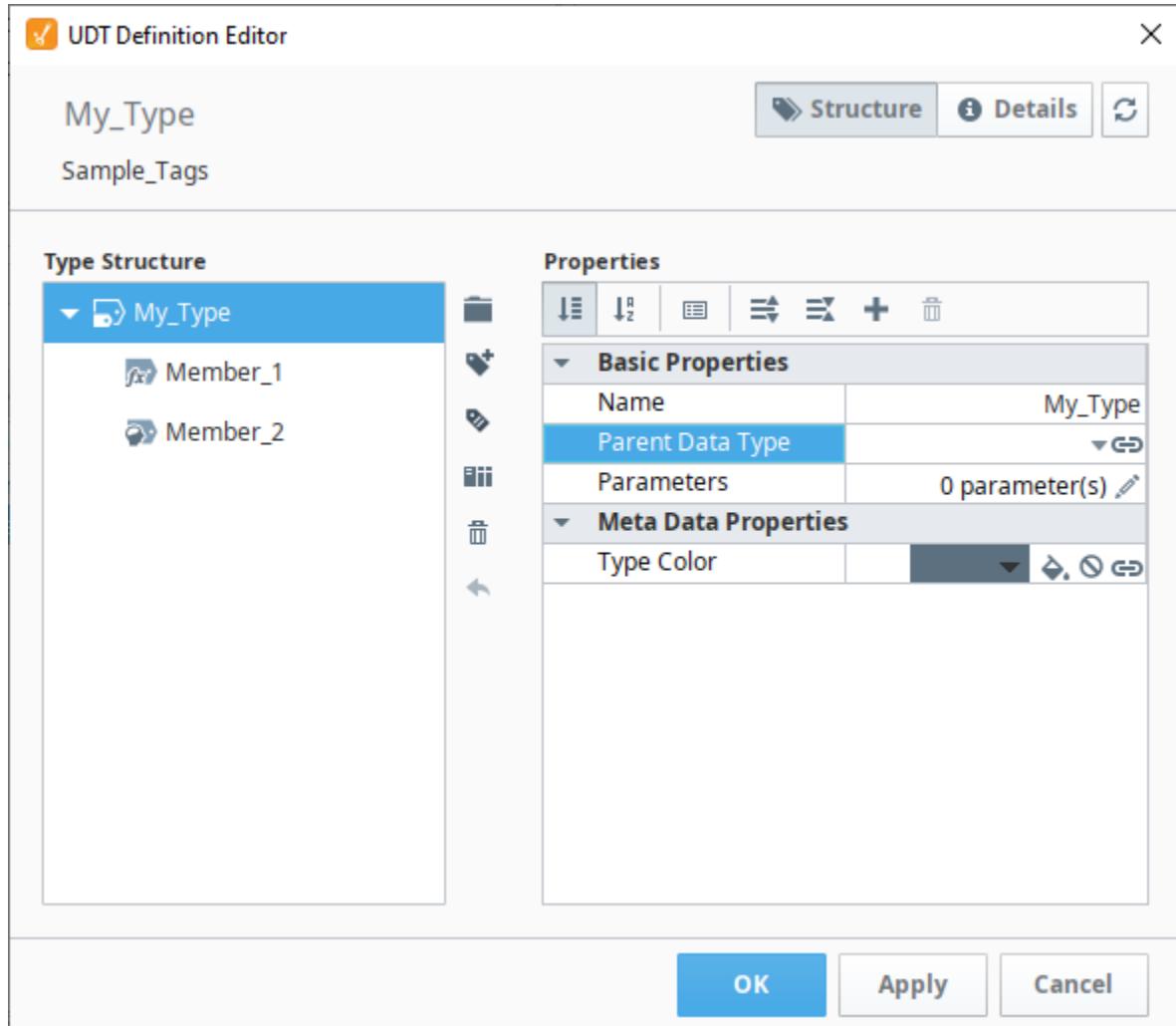


4. Click **OK** to switch back to our instances.. You'll see they now both have new members.



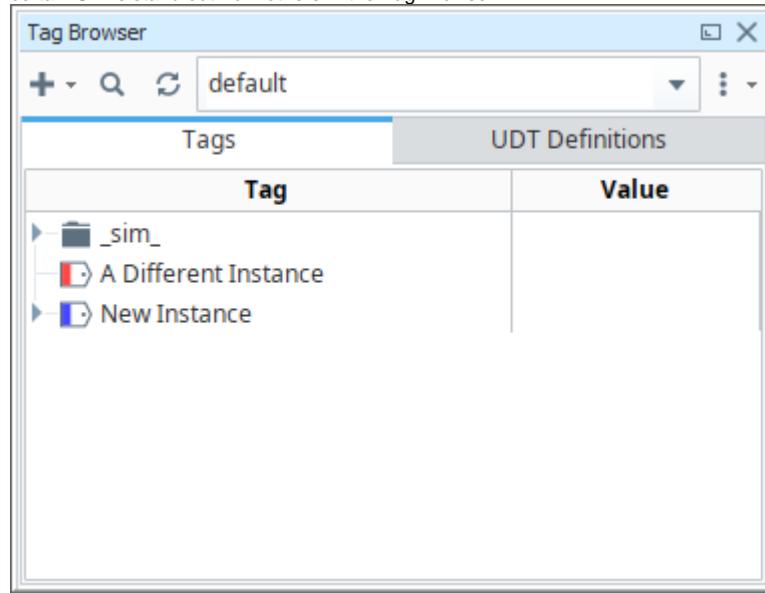
UDT Root Node Properties

While editing a UDT Definition or Instance, the Tag Editor will show some unique properties on the Root Node.



Property	JSON Name	Description
Name	name	The name of the UDT Definition.
Parent Data Type	typeId	<p>Both Instances and Definitions have this property, but the implications of the property are different.</p> <p>On a Definition - The name of the UDT Definition that this Definition is inheriting from. If blank, then the UDT being edited does not inherit from another UDT.</p> <p>On an Instance - The name of the UDT Definition this UDT is an instance of. Changing the Parent Data Type of an instance is not supported.</p>
Parameters	parameters	A collection of parameters configured on the Definition. Note that you can only add or remove parameters on Definitions.
Type Color	typeColor	<p>This feature is new in Ignition version 8.1.0</p> <p>Click here to check out the other new features</p>

A color that will be applied to the Definition and any Instances. This property is only cosmetic, but can be useful to have certain UDTs stand out from others in the Tag Browser.



This feature is new in Ignition version **8.1.0**
[Click here](#) to check out the other new features

Assigning Colors to UDTs

UDTs can be color coded, which applies a color to the Root Node. This is purely a cosmetic change, but can be helpful in systems with a large number of instances, as the colors can make certain UDTs stand out from one another.

Color is applied to the Definition, via the **Type Color** property.

The screenshot shows two windows: the Tag Browser and the UDT Definition Editor.

Tag Browser: The title bar says "Tag Browser". The search bar contains "Sample_Tags". The tabs are "Tags" and "UDT Definitions". The "Tags" tab is selected, showing a tree view with "My Definition" expanded, revealing "Type_A" and "Type_B". A red arrow points from the "Type_A" node in the tree to the "Type Color" property in the UDT Definition Editor.

UDT Definition Editor: The title bar says "UDT Definition Editor". The project name is "Type_A" and the source is "Sample_Tags". The tabs are "Structure" (selected), "Details", and "Copy".

Type Structure: Shows a tree with "Type_A" expanded, containing "Member_1" and "Member_2".

Properties:

- Basic Properties:**
 - Name: Type_A
 - Parent Data Type: (highlighted)
 - Parameters: 0 parameter(s)
- Meta Data Properties:**
 - Type Color: (highlighted, set to green)

Buttons: OK, Apply, Cancel.

Any instances of that type will apply the color to the Root Node.

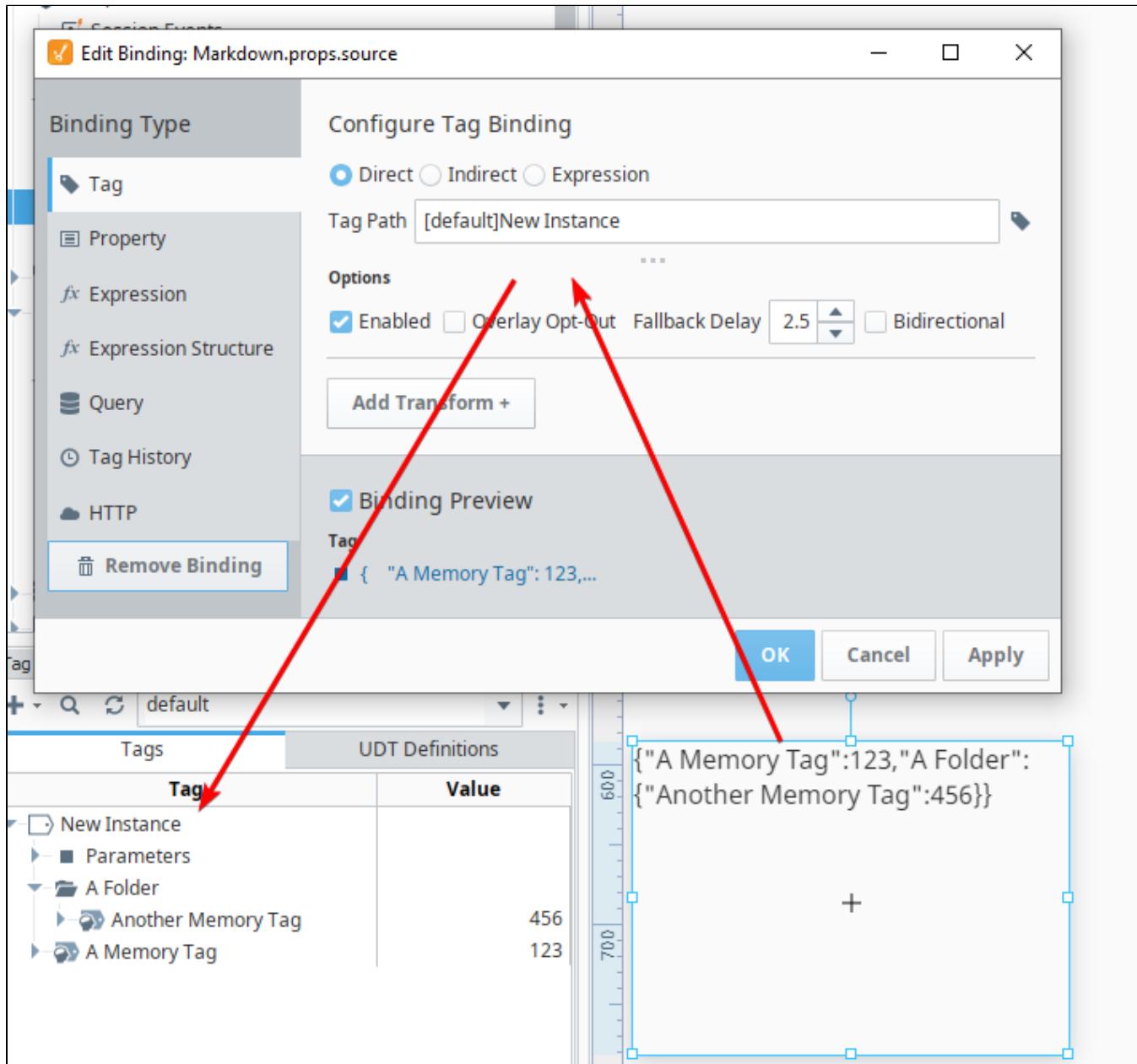
The screenshot shows the Tag Browser.

Tag Browser: The title bar says "Tag Browser". The search bar contains "Sample_Tags". The tabs are "Tags" and "UDT Definitions". The "Tags" tab is selected, showing a tree view with "Ramp" and "An Instance of Type A". "An Instance of Type A" is expanded, showing "Parameters", "Member_1", and "Member_2".

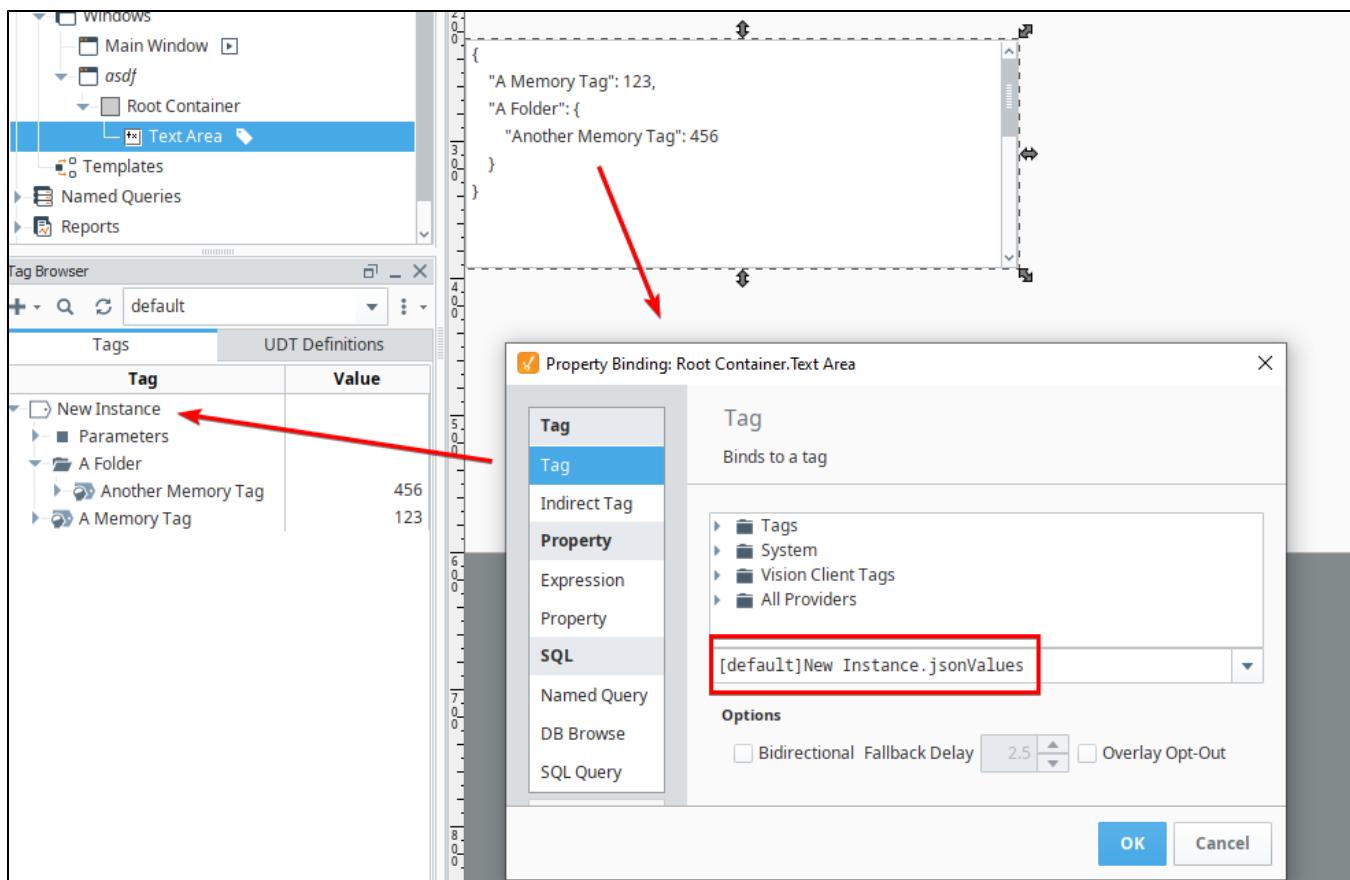
Binding to UDTs

Both Perspective and Vision feature a "binding" system, that allows components in those modules to display live values on Tags. In regards to UDT instances, component bindings can bind to members just like any standard Tag, but they can also bind directly to the UDT instance, which results in the binding receiving live values from all members of the UDT.

In the image below, a Perspective Markdown component has a Tag binding on its **source** property. The binding is leading to the root of a UDT instance. As a result, the member values are shown on the component. They're also live values, so any value changes on any member will appear on the Markdown component.



In Vision, this is also possible, but you'll want to point the Tag binding at the **.jsonValues** attribute on the Tag, which involves manually typing it out in the binding window as follows:



Related Topics ...

- [UDT Parameters](#)

In This Section ...

UDT Parameters

Parameters effectively act as variables that can be referenced by properties on members. A common use case for parameters in UDTs is to make the OPC Item Path on OPC Tag members dynamic, allowing you to replace parts of the OPC Item Path with the parameter's value. However, parameter values can be referenced by other member properties, such as expressions on Expression Tags.

UDT Parameters are configured on UDT Definitions. Instances of a UDT can override the value of a parameter, much like any other property on an Instance.

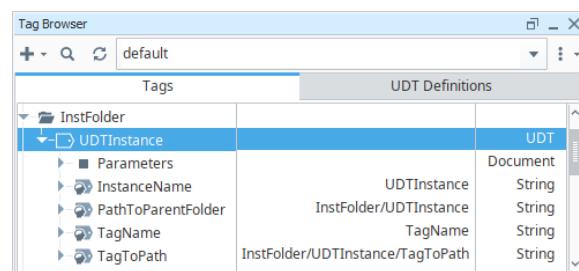
On this page ...

- [Pre-Defined Parameters](#)
- [Adding a Parameter to a UDT](#)
 - [Referencing Parameters from Member Properties](#)
- [Data Type Parameters in Expressions](#)
- [Combining Parameters and Tag References](#)
- [Attribute Referencing and Parameterized Types](#)
 - [Calculations and Numerical Parameter Names](#)

Pre-Defined Parameters

UDTs have a few parameters already defined to make things easier for you. They give you access to the name and various paths associated with a UDT member Tag. These parameters can be accessed from anywhere in a Tag that a normal parameter can be used. Each of these parameters uses that Tag it is in as a starting point for its path.

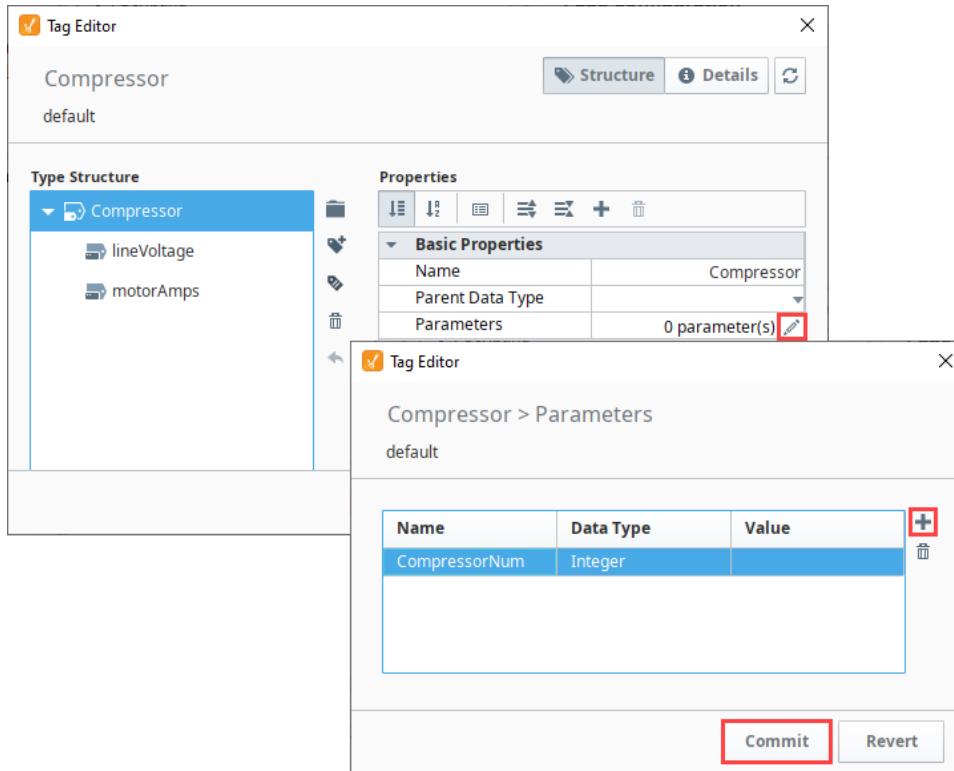
Parameter Name	Description
{InstanceName}	The name of the UDT Instance that this Tag is inside.
{PathToParentFolder}	The full path to the folder that this Tag is in.
{TagName}	The name of the Tag that is using this parameter.
{PathToTag}	The full path to the Tag using this parameter.



Adding a Parameter to a UDT

In this example, our plant has multiple compressors. We created a Compressor UDT, but we want each instance of the Compressor UDT to reference a different set of Tags. Our Compressor UDT has two OPC Tags: lineVoltage and motorAmps. These two Tags are pointing to a specific address in the PLC. In order to reference a different set of Tags for each instance, we need to add a parameter to our Compressor UDT that we called "CompressorNum."

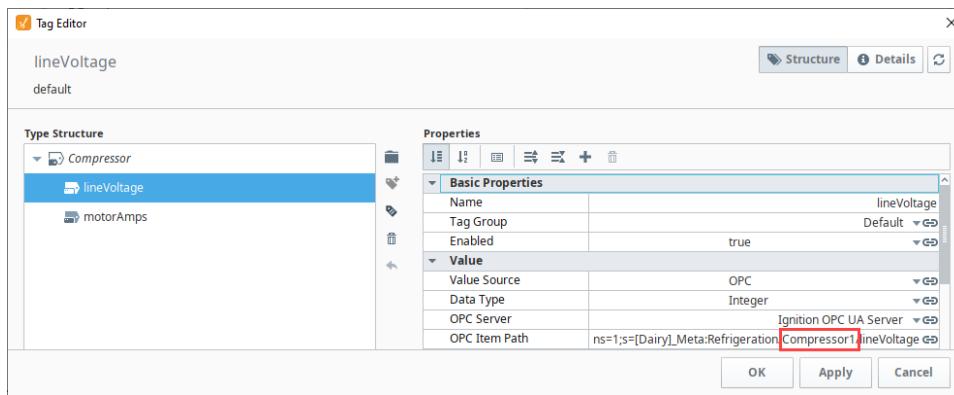
1. In the Tag Browser, we clicked on the **UDT Definitions** tab to find our Compressor UDT.
2. Double click on the Compressor UDT to open it in the Tag Editor and click on the pencil next to the **Parameters** property. The Parameters window will open.
3. Click the Add icon to create a parameter. Enter the parameter **Name** and Data Type of **Integer** and click **Commit**.



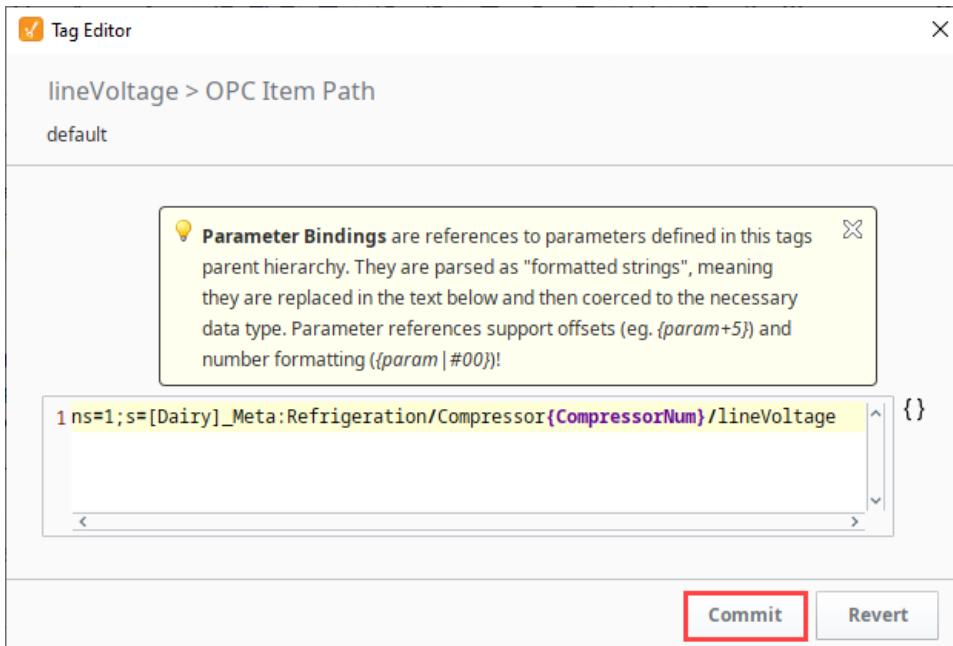
Referencing Parameters from Member Properties

Continuing with the example from above, this next example will show you how parameters are referenced from member properties.

1. In the Tag Editor, let's replace the Compressor Number for each Tag with the new parameter, **CompressorNum**. Select the **lineVoltage** Tag and click on the binding icon and click **Edit**.



2. This opens the OPC Item Path window for editing the **lineVoltage** Tag. Place your cursor at the end of '**Compressor1**', delete the '**1**', and enter '**{CompressorNum}**'. Click **Commit**.



3. Repeat Step 2 for the **motorAmps** Tag.
4. Both Tags will now show the **CompressorNum** parameter in the OPC Item Path.
5. Now, let's create an instance of the Compressor UDT using the CompressorNum parameter under the **Tags** tab of the Tag Browser. To keep UDT instances organized, we created a Plant Compressors folder.
6. Right click on the Plant Compressors folder and select **New Tag > Data Type Instance > Compressor UDT**.
7. Enter a **Name** for the Instance (i.e., HVAC Compressor), then click on the pencil icon next to the **Parameters** property and enter a value (i.e., 2) and press **Commit**. Click **OK** to save the HVAC Compressor instance.

Type Structure

- HVAC Compressor
 - lineVoltage
 - motorAmps

Properties

Basic Properties	
Name	HVAC Compressor
Enabled	true
Parent Data Type	Compressor
Parameters	1 parameter(s)

HVAC Compressor > Parameters

Name	Data Type	Value
CompressorNum	String	2

Commit **Revert**

8. Under the **Tags** tab, you'll see the HVAC Compressor was created showing the Parameter that was used and the values for the OPC Tags listed in the OPC Item Path.

Tag Browser

default

UDT Definitions

Tags	UDT Definitions
Motors 1	
Plant Compressors	HVAC Compressor Parameters CompressorNum String 2 lineVoltage Integer -126 motorAmps Integer 36
Plant Motors	

Data Type Parameters in Expressions

It is possible to use the value of data type parameters directly in expression bindings within a UDT. Parameter references can be quickly inserted into an expression.

While a UDT member is selected in the Tag Editor, you can edit bindable properties, such as the Expression on Expression Tags by clicking the **Edit** icon next to the property.

Tag Editor

Member Number

default

Type Structure

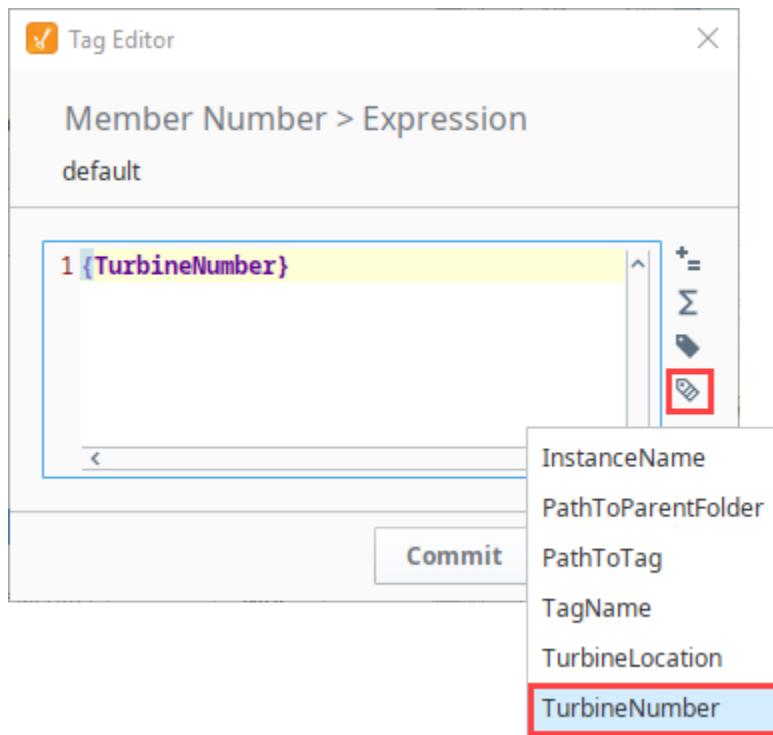
- Turbine
 - Member Location
 - Member Number

Properties

Basic Properties	
Name	Member Number
Tag Group	Default
Enabled	true
Value	
Value Source	Expression
Data Type	Integer
Expression	{TurbineNumber}

OK Apply Cancel

This opens the Expression window. Click on the **UDT Parameters** icon on the right of the expression area, select a parameter, then click **Commit**



Combining Parameters and Tag References

Because parameter and Tag references differ in syntax, some consideration must be made when attempting to use both in the same expression. Tag references must not be placed inside of quotes. After adding a string Tag to the Turbine UDT, a reference to the Tag can be added to Member Location's expression. Single quotes were added to create a space between the Member's Location and the string value.

The screenshot shows the Tag Editor interface with two windows open. The main window displays the 'Type Structure' for a 'Turbine' type, with 'Member Location' selected. The properties for 'Member Location' are shown in the 'Properties' panel, including Name (Member Location), Tag Group (Default), Enabled (true), Value Source (Expression), Data Type (String), and Expression ({TurbineLocation} + '...'). A secondary window titled 'Tag Editor' shows the expression for 'Member Location > Expression' as '1 {TurbineLocation} + ' ' + {[.]String Tag}'. Buttons for 'Commit' and 'Revert' are at the bottom.

Here's what it looks like in the Tag Browser.

The Tag Browser window shows the 'Tags' tab. Under the 'Turbines' folder, there is a 'Turbine 100' folder containing a 'Parameters' folder. Inside 'Parameters' is a 'Member Location' tag. The tag details are as follows:

Tag	Description	Type
Member Location	Folsom, CA This is a string value	String
Member Number	100	Integer
String Tag	This is a string value	String
Wind Speed		Boolean

Attribute Referencing and Parameterized Types

As mentioned above, many properties in the member Tag configuration can reference the parameters available in the data type. When instances are created, these references are replaced with the values defined for the type. Parameter references also support basic offsets and numerical formatting, providing a great deal of flexibility. To reference a parameter, use the syntax `{ParameterName}`.

To offset a value, use the form `{ParameterName+offset}`.

To format a value, use the form `{ParameterName|format}`. The format pattern is the same as that used for the `numberFormat` expression function. In short, "0" can be used to require a digit, and "#" can be used for optional digits. ie: `##0`

Example:

For this example, we'll assume that we're parameterizing the **OPC Item Path**, and that the data type has an integer attribute named `BaseAddress` defined. We'll pretend the **OPC Server** provides Tags named like `DataPoint1`.

Standard Referencing

OPC Item Path: `DataPoint{BaseAddress}`

Offset

Imagine that our data type had three fields, and these were laid out sequentially in the device. Instead of specifying each address for each Tag, we can simply offset from the base address:

Member 1: `DataPoint{BaseAddress+0}`
Member 2: `DataPoint{BaseAddress+1}`
Member 3: `DataPoint{BaseAddress+2}`

Formatting (with offset)

Continuing from the example above, imagine that our OPC server actually provided addresses in the form `DataPoint001`, in order to stay consistent up to "DataPoint999". This can be accommodated using number formatting in the reference:

Member 1: `DataPoint{BaseAddress+0|000}`
Member 2: `DataPoint{BaseAddress+1|000}`
Member 3: `DataPoint{BaseAddress+2|000}`

This format of three zeros means "three required digits". If our instance has a base address of 98, the resulting paths will be `DataPoint098`, `DataPoint099`, `DataPoint100`.

Parameters support more mathematical operators in addition to offsets and formatting. There is a simple expression language available that can be used in conjunction with formatting. The following table shows all available operators in their order of operations (they are evaluated starting at the top of the table).

Operator	Description	Example
<code>()</code>	Parenthesis. These operators are used for grouping any number of values. Also used to change the order of operations.	<code>{Baseaddress*(2+3)}</code>
<code>^</code>	Power. This operator is used to raise a number to a power.	<code>{BaseAddress^2}</code>
<code>-</code>	Negative. Used to create a negative value from one number.	<code>{BaseAddress^-2}</code>
<code>*</code>	Multiplication. Multiply two numbers.	<code>{BaseAddress*2}</code>
<code>/</code>	Division. Dividing the first number by the second number.	<code>{BaseAddress/2}</code>
<code>%</code>	Modulus. This operator returns the remainder of a division operation. IE: $7/3 = 2$ with a remainder of 1, so $7\%3 = 1$	<code>{BaseAddress%2}</code>
<code>+</code>	Addition. Add two numbers.	<code>{BaseAddress+2}</code>
<code>-</code>	Subtraction. Subtract two numbers	<code>{BaseAddress-2}</code>

Example

```
# This dynamic OPC Item path takes in three parameters to determine the tag path
ns=1;s=[DeviceName]Path/to/tag{BaseAddress+(ParamNum*Multiplier)|0000}

# The OPC Item path resolves to the following assuming the following values:
# BaseAddress = 5
# ParamNum = 8
# Multiplier = 2
ns=1;s=[DeviceName]Path/to/tag0021
```

Calculations and Numerical Parameter Names

If the parameter names are purely numerical values (we don't recommend this: it gets confusing), then quotation marks must encase the parameter to run any sort of calculations on the value of the parameter.

For example, if a UDT contains a parameter named 0, and its value is 10:

```
// This will evaluate to 0, because it thinks you mean the integer 0, not the parameter named "0"  
{0 * 1000}  
  
// This will evaluate to 10000, because the quotation marks denote a parameter named "0"  
{"0" * 1000}
```

Related Topics ...

- [UDT Nesting](#)
- [Expression Language and Syntax](#)

UDT Multi-Instance Wizard

The Multi-Instance Wizard provides a powerful, but simple mechanism for rapidly generating many instances of a UDT at the same time by specifying patterns for UDT parameters.

Value Patterns and Tag Names

Value Patterns

In order to define values for parameters (and the Tag names), you can use several different types of patterns (and combinations of patterns):

Range number1-number2[/step]

A numeric range of values, such as 1-10. Optionally, a step parameter can be included, in order to only generate numbers at certain multiples. For example, 0-100/10 would generate 0, 10, 20, and so on.

Repeat value*count

A value (numerical or string), and the number of times to use it. For example, North Area*10 would use the parameter North Area for 10 items.

List value1, value2, value3

A comma separated list of values (or other patterns) to use.

Examples:

1-10,21-30,31-40 Results in 30 Tags being created, with the specified value ranges (so, for example, there would be no parameter 15).

A,B,C Results in 3 Tags, with each of the values.

0-100/5 Results in 21 Tags (because range is inclusive), with values 0, 5, 10...100.

As mentioned, the size of the pattern will dictate how many Tags will be created. If some patterns are smaller than others, the last value will be repeated for the other Tags.

Tag Names

The names of the generated instances can be specified using a system similar to that of the parameter patterns. If you just want to use sequential names, you don't need to specify a pattern, as values will be generated automatically starting at one. You can also set the pattern to simply be the starting number to generate sequential names from there.

Base Name

A string base for the Tag name. This can also be a list of names, in which case the names will be used directly, and the name pattern won't be used.

Name Pattern

A pattern that will be used to generate values that will be appended to the base name.

At any time, you can use the **Preview** button to view the Tag names and parameters that will be created. Once you are satisfied, click **OK** to generate the Tags under the selected folder in the Tag provider.

How to Make New Instances of a UDT

Once you have a UDT Definition created, you can make multiple instances of it with the Multi-instance Wizard.

1. In the **Tag Browser**, click the Add  icon to create a new folder. We created one called Machine Motors.
You can create a single instance of the motor or use the Multi-instance Wizard to rapidly create many instances at the same time. In this example, we will use the Multi-Instance Wizard.
2. Right-click on the **Machine Motors** folder, and select **Multi-instance Wizard** to open the Instance Creation Wizard window.

On this page ...

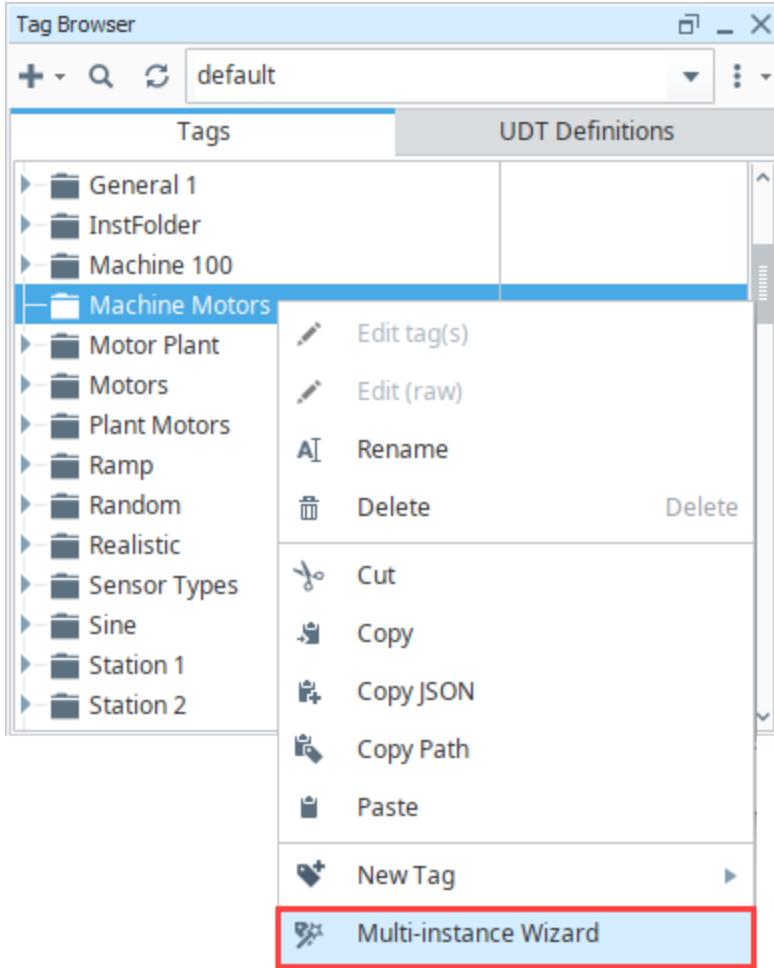
- [Value Patterns and Tag Names](#)
 - [Value Patterns](#)
 - [Tag Names](#)
- [How to Make New Instances of a UDT](#)



INDUCTIVE
UNIVERSITY

UDT Multi-Instance Wizard

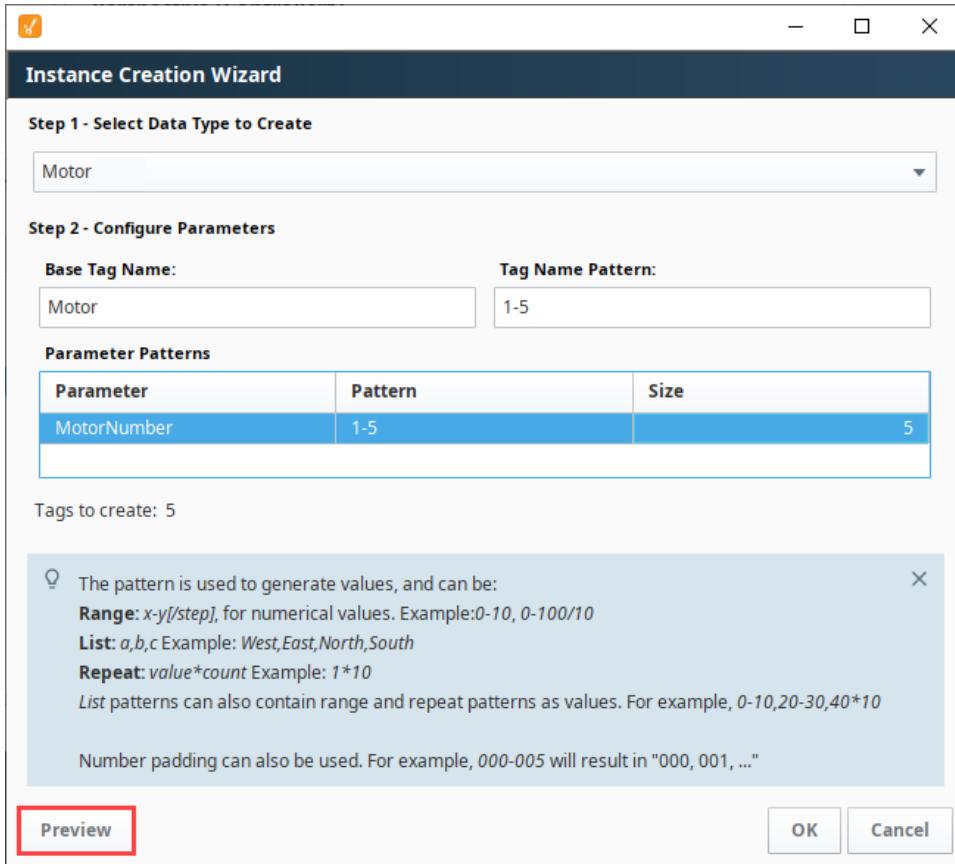
[Watch the Video](#)



3. In **Step 1 - Select Data Type to Create**, select a UDT (i.e., **Motor**) from the dropdown.
4. In **Step 2 - Configure the Parameters**, enter the following for your Motor:

- Base Tag Name: "Motor." Note the space at the end. Without this space your Tag names will look like Motor1, Motor2, etc.
- Tag Name Pattern: 1-5 This creates five Tags **Motor 1**, **Motor 2**, thru **Motor 5**.
- Parameter Patterns: the **MotorNumber** parameter is entered by default when we selected the data type to create in Step 1.
- Pattern: **1-5** is the pattern of the parameter so the Motor 1 Tag will have a parameter of 1, Motor 2 will have a parameter of 2, and so on through Motor 5.

You'll notice that after you enter the Pattern, the number of Tags to create is updated. In this example, five Motor Tags will be created. Click **Preview**.



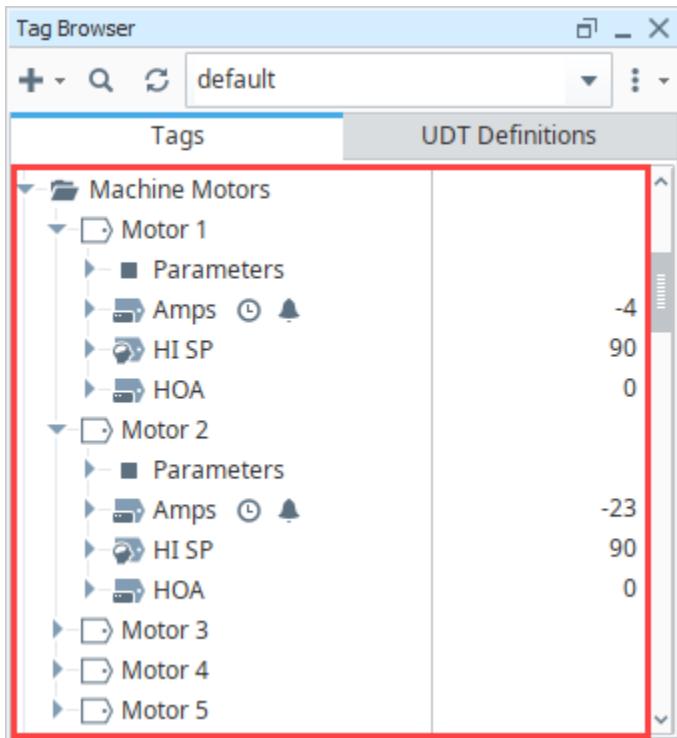
5. In Preview, you will see how the Base Tag Names and Parameter Values get created. Click **Back** to go back to the Instance Creation Wizard window if you want to make an update. If you like what you see on the Preview window, click **OK**.

The screenshot shows the 'Instance Creation Wizard' window in 'Preview' mode. It displays a table with two columns: 'Tag Name' and 'MotorNumber'. The rows show the following data:

Tag Name	MotorNumber
Motor 1	1
Motor 2	2
Motor 3	3
Motor 4	4
Motor 5	5

At the bottom are 'Back' (highlighted with a blue box), 'OK' (highlighted with a red box), and 'Cancel' buttons.

6. In the Tag Browser, expand Motor Tags 1-5 to see if all the members of the Motor UDT were created and are running.



Cannot Edit Existing Instances using the Multi-Instance Wizard

You cannot edit existing instances using the Mult-Instance Wizard. The Mult-Instance Wizard is only used for quickly creating many instances of a UDT at the same time. If you want to make a change to all your instances, refer to [How to Edit an Existing UDT](#).

Related Topics ...

- [UDT Parameters](#)

UDT Inheritance

Once you have a single data type created, it is possible to set up UDT inheritance where data types extend to other data types, to add additional members, or override default values. For example, you can create a new data type and using the inheritance feature it will inherit all Tags from the parent data type including the parameters. Then you can add additional Tags and/or override any settings in your new data type. UDT Inheritance is a way to extend to a class of data types to add more functionality to that class.

For example, you may have a simple motor and a complex motor. The complex motor can inherit from the simple motor, which means all simple motor values will be in the complex motor and you can add more.

Nesting (using one or more UDTs to make up a larger UDT) is different from inheritance and can be found under [UDT Nesting](#).

On this page ...

- [To Inherit Property Values from an Existing UDT](#)
- [Creating the Data Type Instance](#)
- [Overriding Properties of the Parent UDT](#)
- [UDT Inheritance Traits](#)



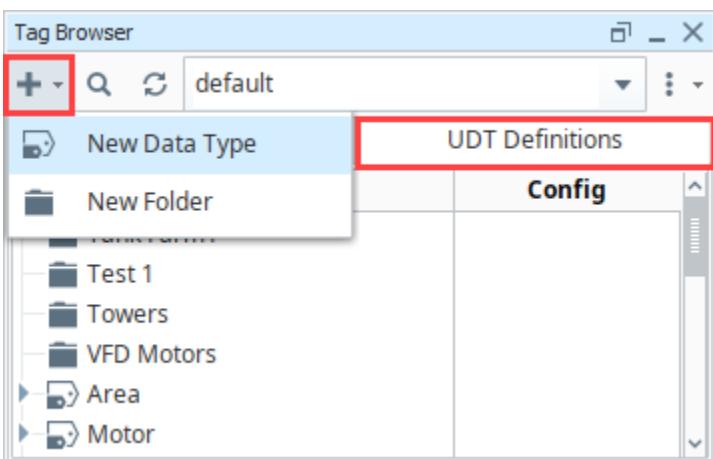
UDT Inheritance

[Watch the Video](#)

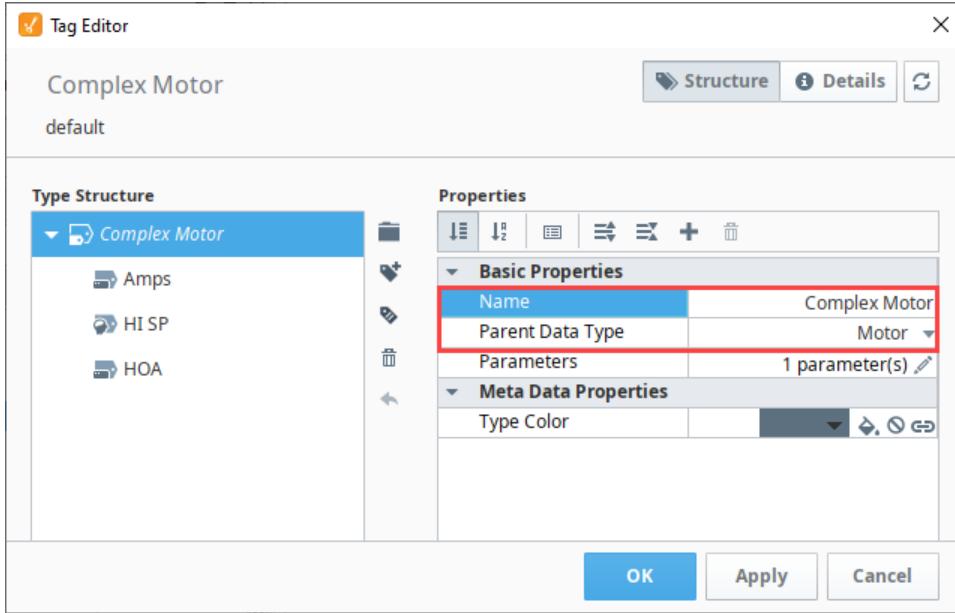
To Inherit Property Values from an Existing UDT

Let's use our data type Motor from the previous sections to create another data type. We'll set the parent to Motor so our new data type automatically inherits all the properties of Motor.

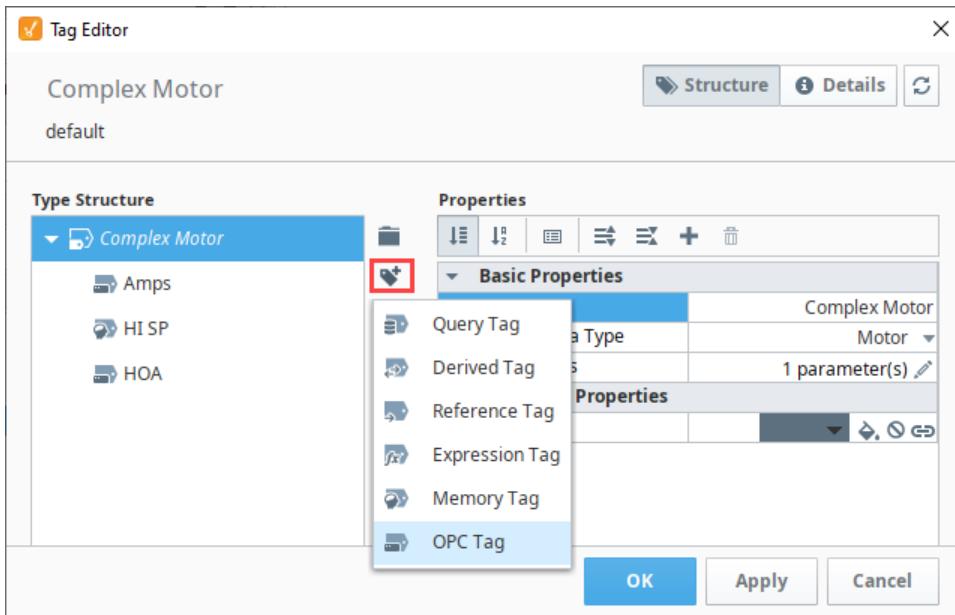
1. In the **Tag Browser**, click on the **UDT Definitions** tab and click the Add icon to create a **New Data Type**. The Tag Editor window will open on the.



2. In the **Name** field, enter name for your new UDT (i.e. Complex Motor). Under **Parent Data Type** property go to the dropdown and select the **Motor** data type and click **Apply**. Now your new Complex Motor UDT is inheriting from all the properties of the parent Motor UDT: Amps, HI SP and HOA.



3. With the Tag Editor still open, let's add a **OPC Tag** to the Complex Motor UDT. Click on the **Add Tag** and select **OPC Tag**.



4. Enter the following properties for your new Tag and click **Apply**. You will see the new Tag was added to the Complex Motor UDT.

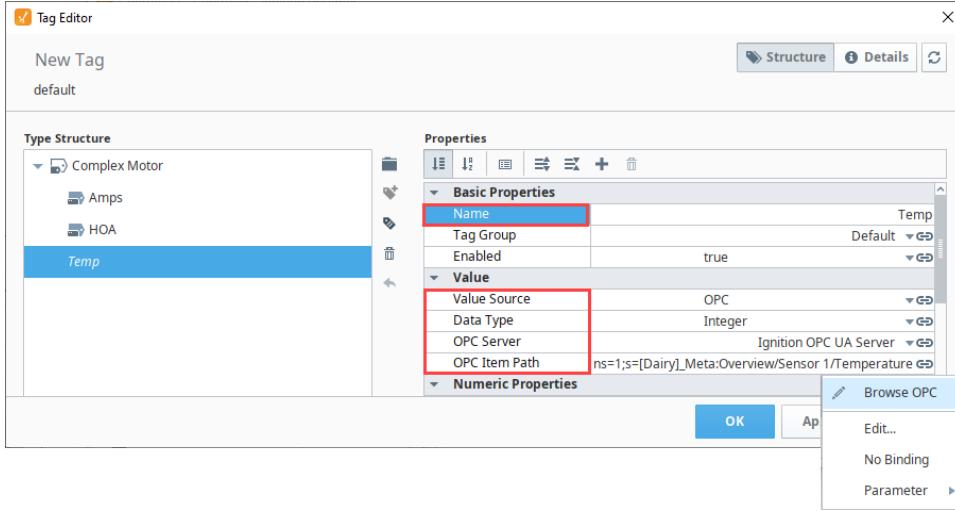
Name: Temp

Value Source: OPC

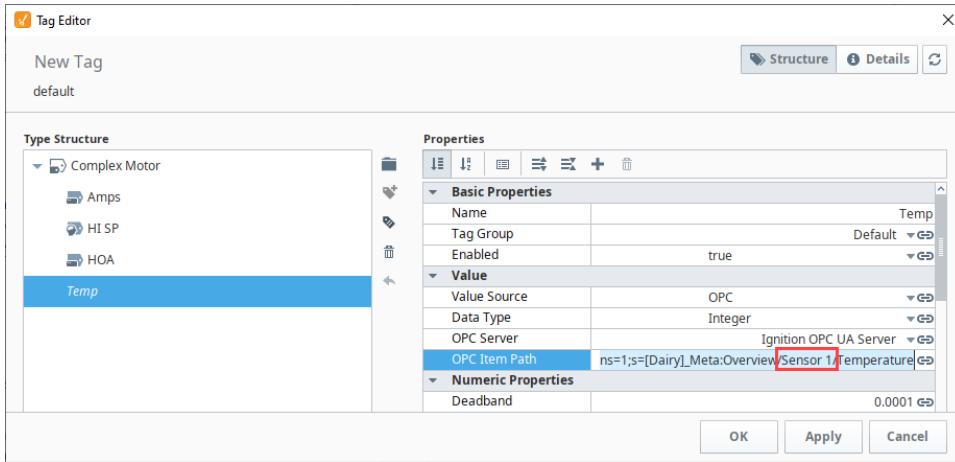
Data Type: Integer

OPC Server: Click on the binding () icon and select **Ignition OPC UA Server**

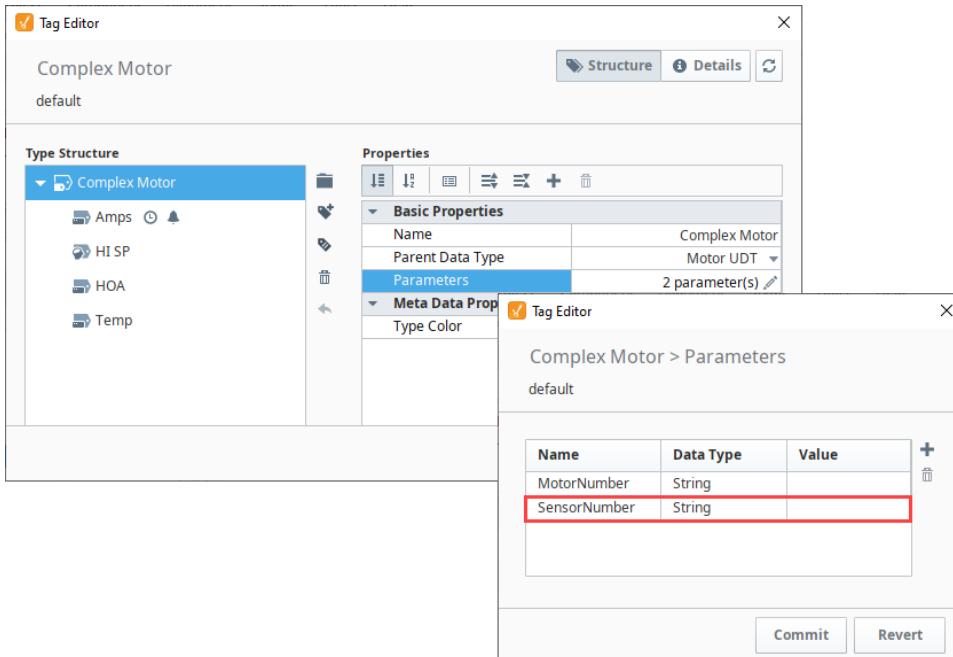
OPC Item Path: Browse the [OPC](#) and find the Tag you want to use. This example uses a **Temperature** Tag from a Sensor in the Dairy program. Click **Commit**.



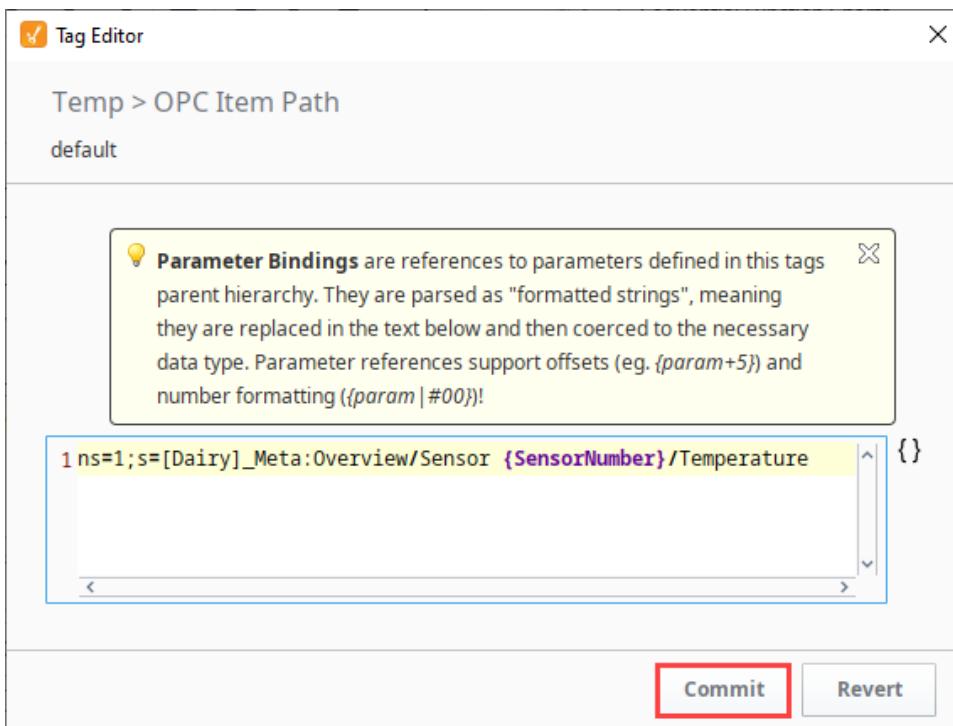
5. In the image below the Temp Tag is pointing to a specific address in the PLC. Because we're creating a new Tag in our UDT, we don't want to point to one specific set of **'Temp'** Tags. We want each instance of the Complex Motor UDT to reference a different set of **'Temp'** Tags. To do that, we need to add a parameter to the Complex Motor data type that we will call **'SensorNumber'**.



6. With the Complex Motor UDT **Tag Editor** open, let's create a new UDT parameter. Right click the **Edit** icon next to the **Parameters** property. The Parameters window will open.
7. Click the Add **+** icon and add the new parameter, **'SensorNumber'** and click **Commit**.



8. With the Tag Editor still open, select the "Temp" Tag. In the **OPC Item Path** field, click the binding icon, select **Edit**, and the **Temp > OPC Item Path** window will open. Place your cursor at the end of 'Sensor1', delete the '1', add a space, and enter '{SensorNumber}'. Don't forget the curly braces. The OPC Item Path will look like the following image, then click **Commit** to save your updates and go back to the previous window.



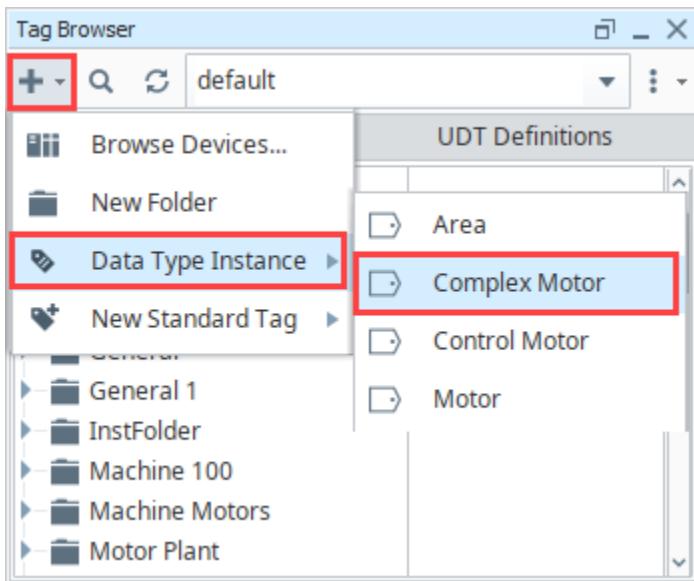
9. In the **Tag Editor** window, click **OK** to save your UDT. In the Tag Browser, the new data type is now visible in the "Complex Motor." You can see all the inherited Tags from the Motor data type and the newly added Temp Tag.

Tags		UDT Definitions
Definition	Config	
Tank Farm1		
Test 1		
Towers		
VFD Motors		
Area		
Complex Motor		
Amps	Ignition OPC UA Ser...	
HISP	90	
HOA	Ignition OPC UA Ser...	
Temp	Ignition OPC UA Ser...	
Control Motor		
Motor		

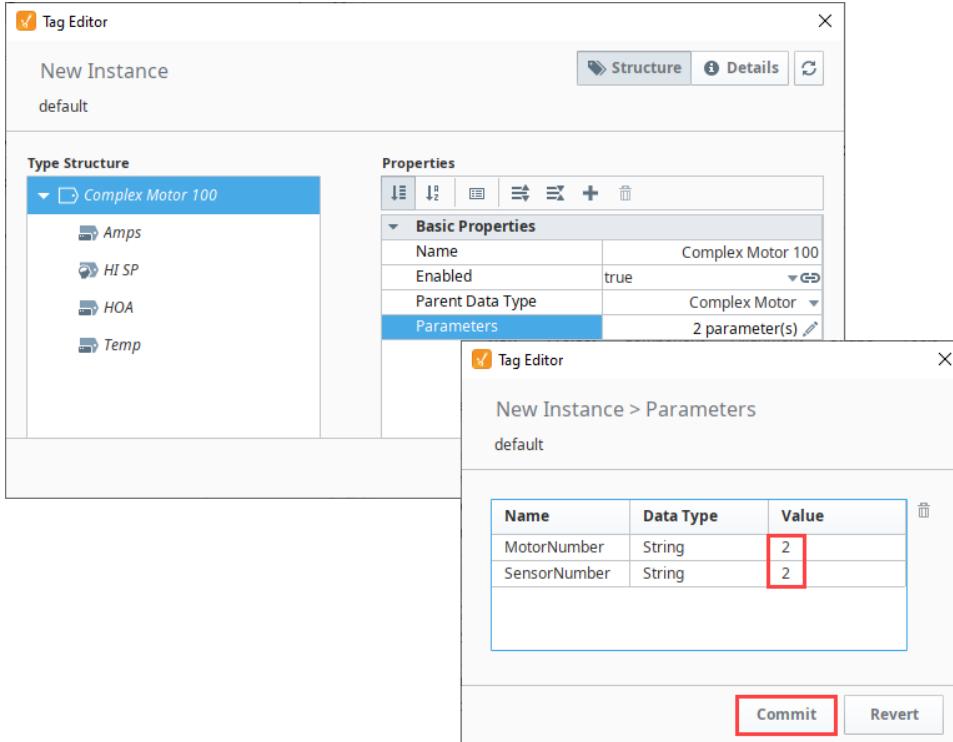
Creating the Data Type Instance

Now that our Complex Motor UDT is set up, let's create a data type instance of the Complex Motor.

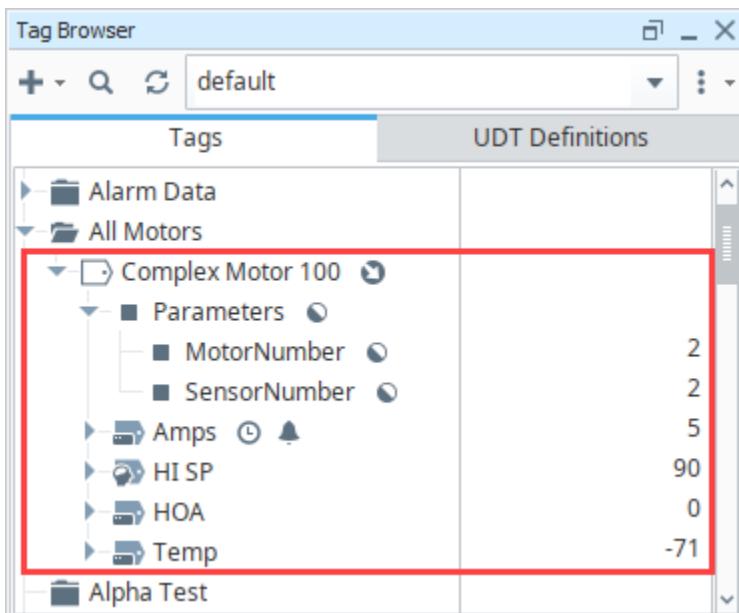
1. Click on the **Tag** tab, and from the context menu, select the **Data Type Instance > Complex Motor**.



2. Enter the **Name** for the instance (i.e., Complex Motor 100). Click the **Edit** icon next to the **Parameters** property and enter the parameter value of '2' for **MotorNumber** and **SensorNumber**, and press **Commit**.



3. Click **OK** to create the instance. Now, you'll be able to see all the values for Complex Motor 100 including the Temp Tag that was added.



Overriding Properties of the Parent UDT

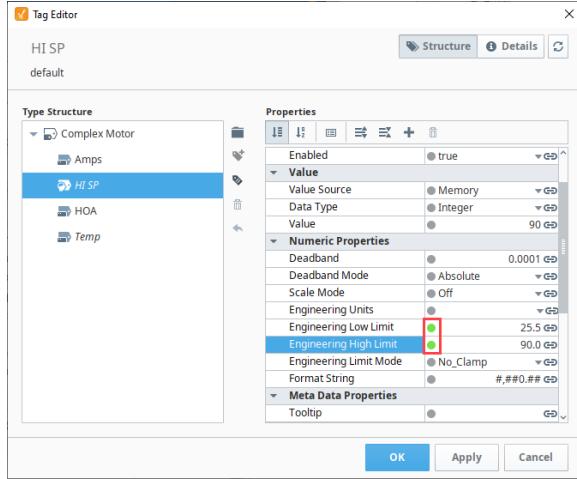
Another benefit of the UDT inheritance feature is it allows you to override some of the properties of the parent. For example, since the Complex Motor has Motor as the parent, you can go to any of the Tags and override any of the settings of that data type. Click the circle to the right of the property and enter a new value, or change a property's value and the green circle changes to green automatically. This



**Overriding
Properties in UDT
Instances**

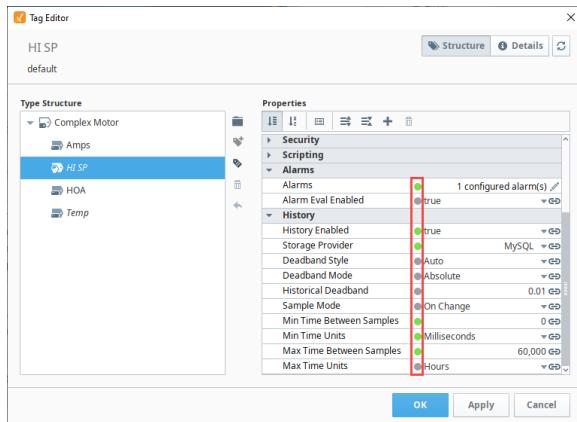
overrides the property inherited from the parent.

[Watch the Video](#)



You can also turn on Alarming and History that wasn't initially turned on in the parent UDT by simply using the override feature. Next to the Alarm property, click the green circle to change it to green, and

click the pencil () icon to configure the alarm if it is not already not configured. If you want to turn on History, click the green circle or change any of the History properties which will cause any of the green circles to change from gray to green.



UDT Inheritance Traits

UDT instances that are inherited and have properties that override the parent properties, have a visual representation next to them.

Tag Browser

default

Tags UDT Definitions

Tags	UDT Definitions
Complex Motors	Complex Motor
Motor 1	Document
Parameters	Integer
MotorNumber	1
SensorNumber	1
Amps	Integer
HI SP	Integer
HOA	Integer
Temp	Integer
Motor 2	Complex Motor
Motor 3	Complex Motor

Related Topics ...

- [UDT Parameters](#)
- [UDT Nesting](#)

UDT Nesting

It's possible to set up UDT nesting in Ignition where you are putting one UDT inside of another UDT. The UDT is nested as an instance within another UDT. It facilitates quicker development of projects since you're able to piece together multiple UDT definitions as needed without having to build everything from the ground up as with each UDT definition. This is particularly useful because it promotes rapid development if you are expanding a plant or facility where all you have to do is make a few Tag changes to existing parameters and property settings.

For example, you may have a production line that is built out of several different machines. You don't need to re-create a motor for each line, instead you can create it once and use it in every line.

Inheritance (having simple and complex version of similar objects) is different from nesting UDTs, and can be found under [UDT Inheritance](#).

On this page ...

- [Set Up UDT Nesting](#)



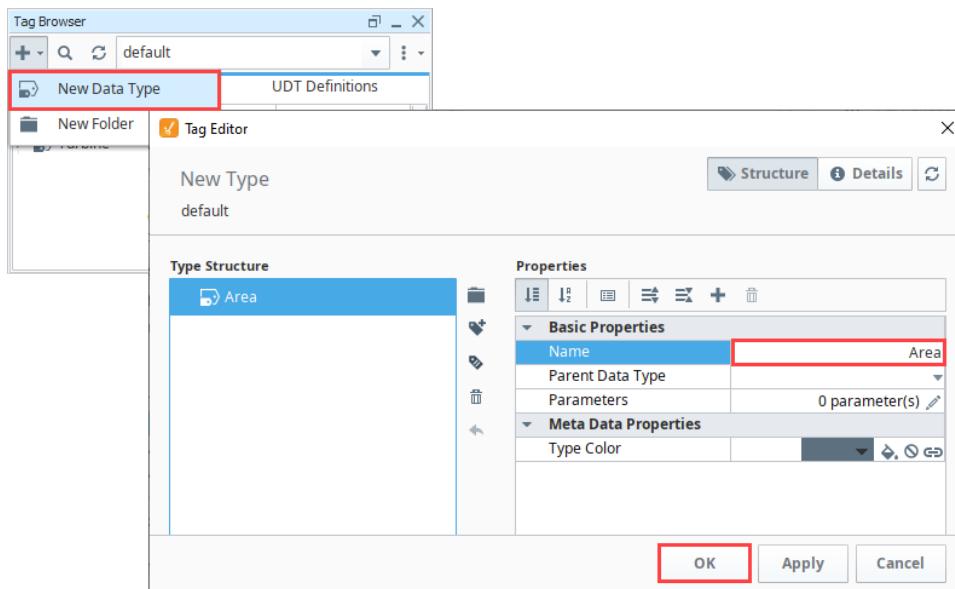
Nested UDTs

[Watch the Video](#)

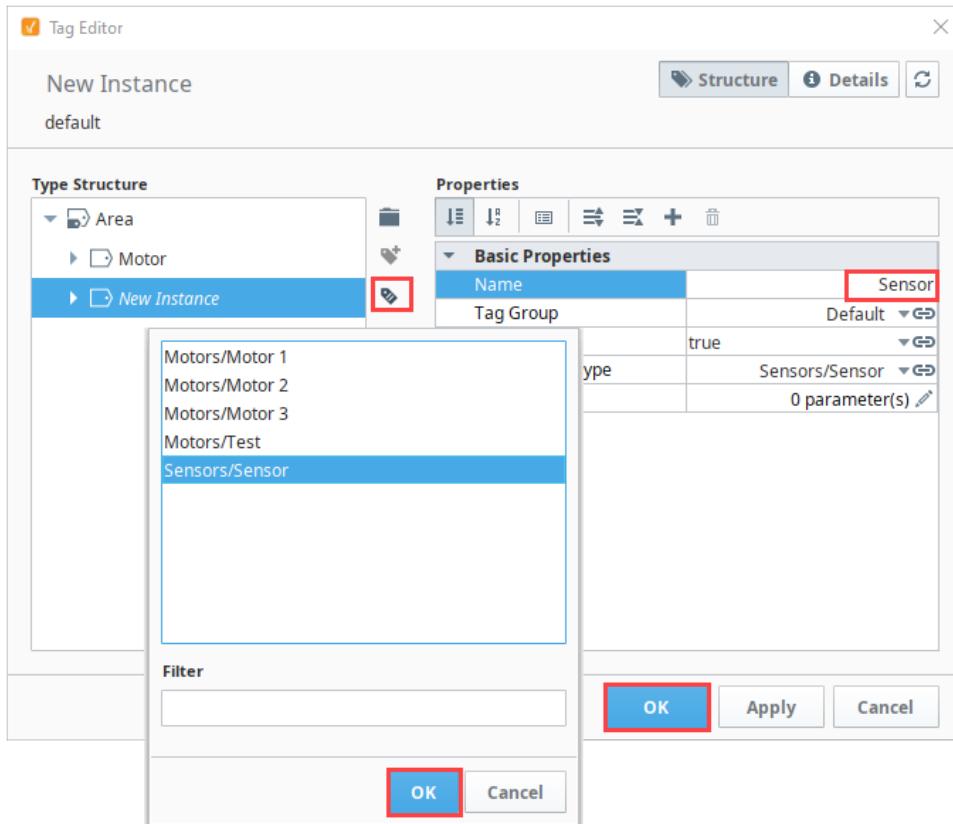
Set Up UDT Nesting

In this example, let's use our a Motor and Sensor data types that were created and used in previous sections of this manual. We are going to create a third UDT called Area that will contain the Motor and Sensor data types inside of it.

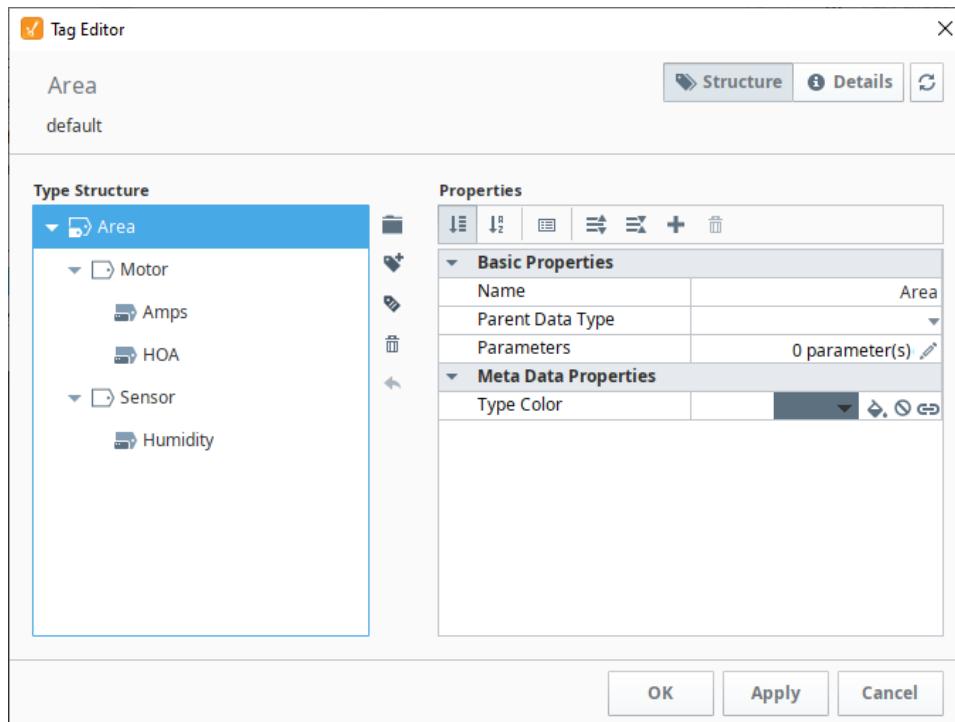
1. In the Tag Browser, right click on **Data Types** and select **New Tag > New Data Type**. The Tag Editor window will open. Assign the new data type a Name called **Area**.



2. Inside of the **Area** data type, create two data type instances, one for Motor and the other for Sensor:
 - a. Click on the **New UDT Instance** icon, select **New UDT Instance > Motor/Motor1**. Click **OK**.
 - b. Rename the new instance '**Motor**'. Click **Apply**.
 - c. Click on the **New UDT Instance** icon, select **New UDT Instance > Sensors/Sensor**. Click **OK**.
 - d. Rename the new instance '**Sensor**'. Click **Apply**.

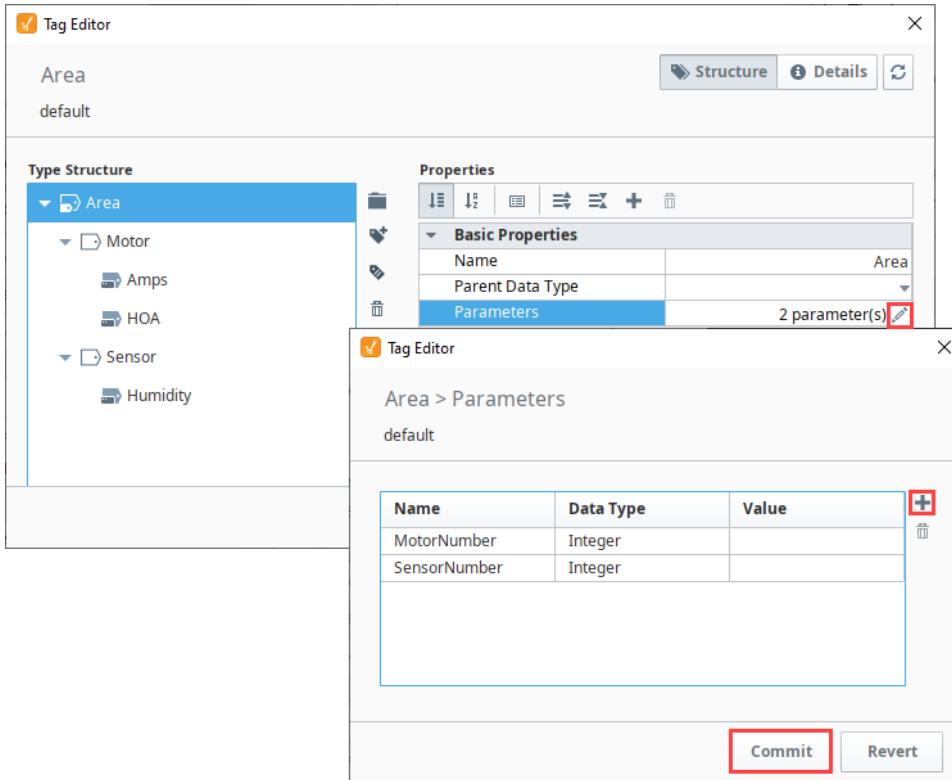


3. With the Tag Editor still open, you'll notice that both the Motor and Sensor UDTs were added. For every UDT that you add inside another UDT, those UDT instances have parameters that need to be specified. In this example, the Motor UDT has the '**MotorNumber**' parameter, and the Sensor UDT has the '**SensorNumber**' parameter. You must pass a value into the each of these UDTs (Motor and Sensor) from the parent UDT (Area). To view the parameters for each UDT instance, select each UDT instance (Motor and Sensor) and click the **Edit** icon.



4. Now that you know what parameters are in each UDT instance, go to the Area UDT, and click the **Edit** icon next to the **Parameters** property. The Parameters window will open.

5. Click on the Add  icon to add the 'MotorNumber' and 'SensorNumber' parameters, then click **Commit**.

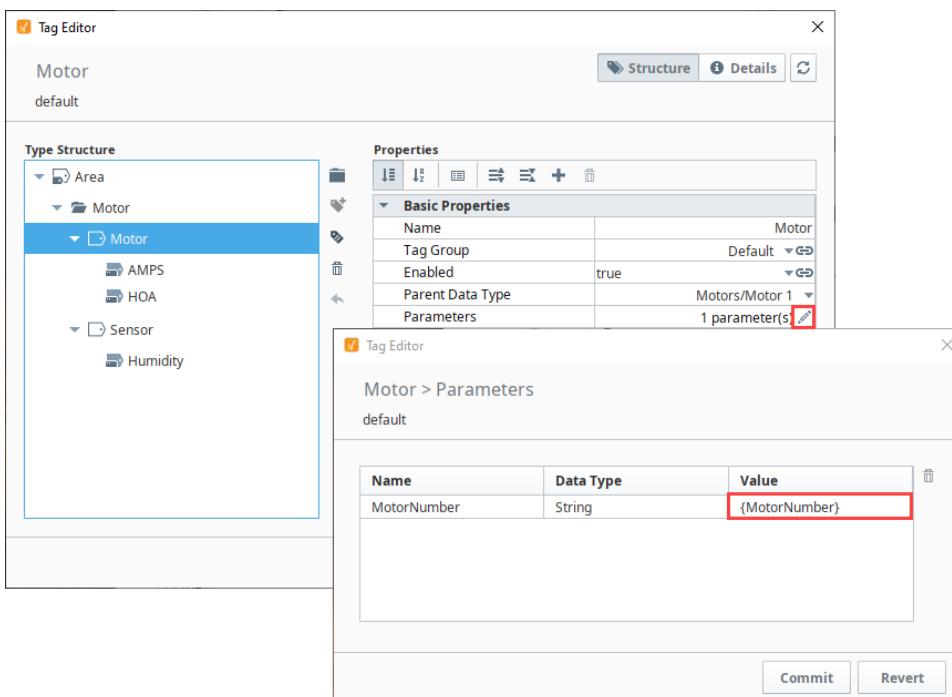


The screenshot shows two windows of the Tag Editor. The left window displays the 'Type Structure' for the 'Area' data type, which contains a 'Motor' folder with 'Amps' and 'HOA' sub-items, and a 'Sensor' folder with 'Humidity'. The right window shows the 'Properties' tab for the 'Area' data type, specifically the 'Parameters' section. It lists '2 parameter(s)' with a red box around the text. Below it is another Tag Editor window titled 'Area > Parameters' showing a table with two rows: 'MotorNumber' (Data Type Integer) and 'SensorNumber' (Data Type Integer). At the bottom of this window are 'Commit' and 'Revert' buttons, with 'Commit' also highlighted by a red box.

6. Now we need to pass these values into the UDT instances by adding a reference. Select the **Motor** UDT, and click the **Edit**  icon next to the **Parameters** property.
 7. Enter the reference for MotorNumber: '{MotorNumber}'. Click **Commit**.

 **If Multiple Data Types use the same Parameter**

In the event data types use the same parameter, you only need to enter it once in the new data type (i.e., Area).



The screenshot shows two windows of the Tag Editor. The left window displays the 'Type Structure' for the 'Motor' data type, which contains a 'Area' folder, a 'Motor' folder (selected), and a 'Sensor' folder. The right window shows the 'Properties' tab for the 'Motor' data type, specifically the 'Parameters' section. It lists '1 parameter(s)' with a red box around the text. Below it is another Tag Editor window titled 'Motor > Parameters' showing a table with one row: 'MotorNumber' (Data Type String) and a value of '{MotorNumber}' (highlighted with a red box). At the bottom of this window are 'Commit' and 'Revert' buttons.

8. Do the same steps to add a reference to the **Sensor** UDT. Select the Sensor UDT and click the **Edit**  icon next to the **Parameters** property. Enter the reference for SensorNumber: '{SensorNumber}'. Click **Commit**.
9. Click **OK**.

Related Topics ...

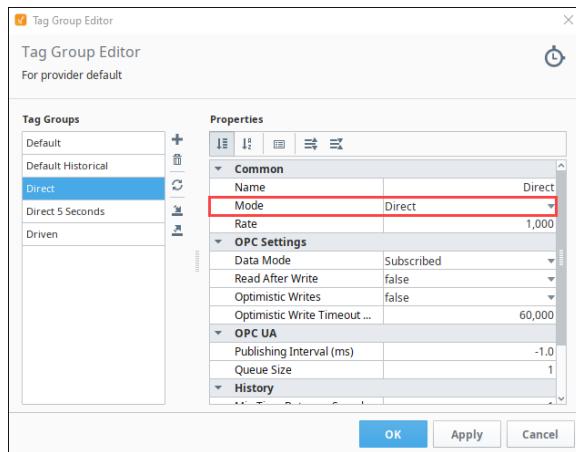
- [UDT Parameters](#)
- [Data Type Parameters in Expressions](#)

Tag Groups

What Is a Tag Group?

Tag Groups dictate the rate of execution of Tags, and therefore play a crucial role in the design of large and high-performance systems. Highly optimized systems generally use multiple Tag Groups which allows for each data point to be polled only as fast as it needs to be. For example some Tags may need to be updated every 500ms, while others may only need to be updated every 30 seconds.

Creating different Tag Groups allows you to organize your Tags into groups that subscribe or poll at different rates.



Tag Groups Modes

Tags are assigned to a Tag Group that determine how often they update. For example, how often an OPC value is polled from the PLC, how often an Expression Tag calculates its expression, and how often a Query Tag runs its query. It's easy to create Tag Groups in Ignition for just about any scenario you can think of. Tag Groups are extremely powerful and flexible, and you can create them based entirely on your individual business requirements.

There are three different Tag Group modes in Ignition that you can use. Each mode works a bit differently.

Direct

Executes at one fixed rate, as determined by the **Rate** property in milliseconds. This is the most simplistic option, as the rate of the group doesn't intentionally change between two different rates. The default tag group on newly installed Ignition systems uses this mode. It's the simplest mode in that there is only a single rate for this mode.

On this page ...

- [What Is a Tag Group?](#)
- [Tag Groups Modes](#)
 - Direct
 - Driven
 - Leased
- [Adding and Editing Tag Groups](#)
- [Setting a Tag Group on a Tag](#)
- [Tag Group Properties](#)



Tag Group Overview

[Watch the Video](#)



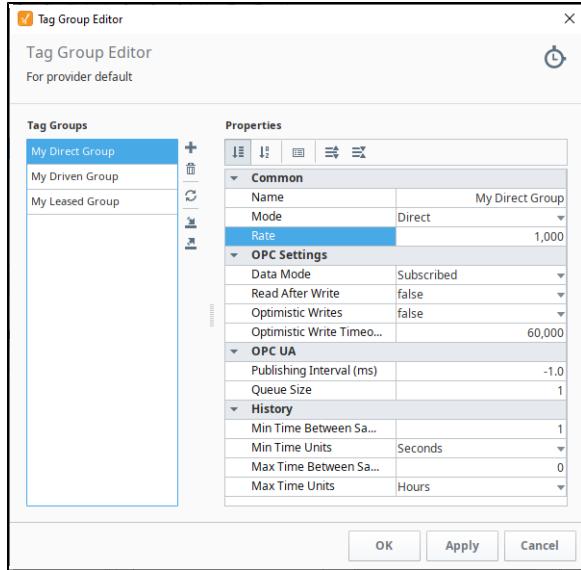
Driven Tag Group

[Watch the Video](#)



Leased Tag Group

[Watch the Video](#)



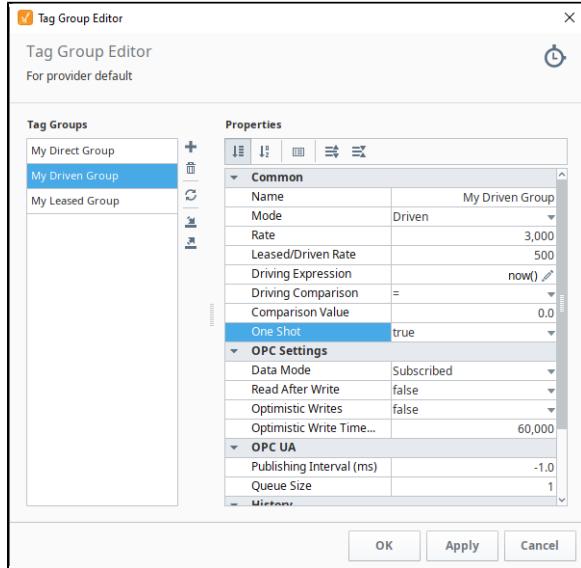
Driven

Driven Tag Groups switch between two different rates based on a condition you configure directly onto the Tag Group. The condition in this case is an [expression](#) on the **Driving Expression** property. The result of the expression is then compared to the **Comparision Value** property. If the comparison of the two values is equivalent, then the entire tag group executes at the **Leased/Driven Rate** in milliseconds, otherwise it executes at the **Rate**.

Thus, you can have a tag group that switches between a fast and slow rate under certain conditions.

Furthermore, Driven Tag Groups have a **One Shot** property. When set to true, means the group only executes every time the comparison is equivalent. Furthermore, it only executes once every time that comparison transitions from false to true: this is sometimes called "a rising edge".

The One Shot property allows for more interesting tag executions. Instead of running at set rates, you can trigger an execution for all tags on the group by incorporating a tag value as part (or all) of the driving expression, allowing a single tag value change to cause many other tags to update.

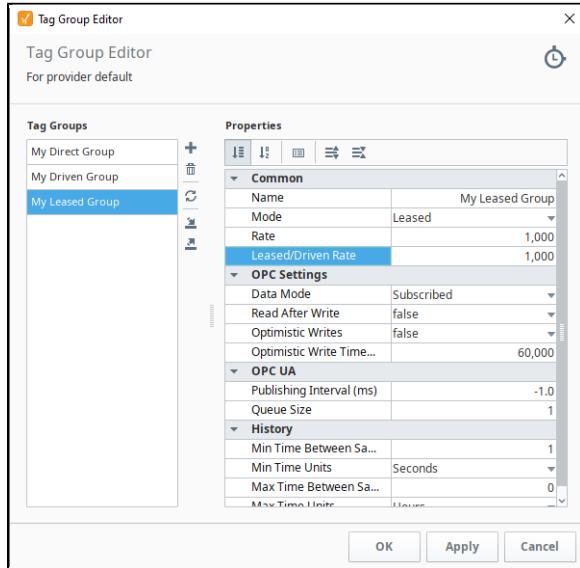


Leased

Leased groups have two different execution rates, much like a Driven group. However Leased groups don't rely on a comparison to determine rate. Instead, the driving mechanism is whether a Tag is being displayed on an open window or session: generally via a Tag Binding of some sort, but being visible in the Tag Browser from a Designer session also counts as being "displayed".

Tags that are displayed will run at the fast rate (**Leased/Driven Rate** value), while those Tags on the same Leased group that aren't displayed will run at the slow rate (**Rate** value).

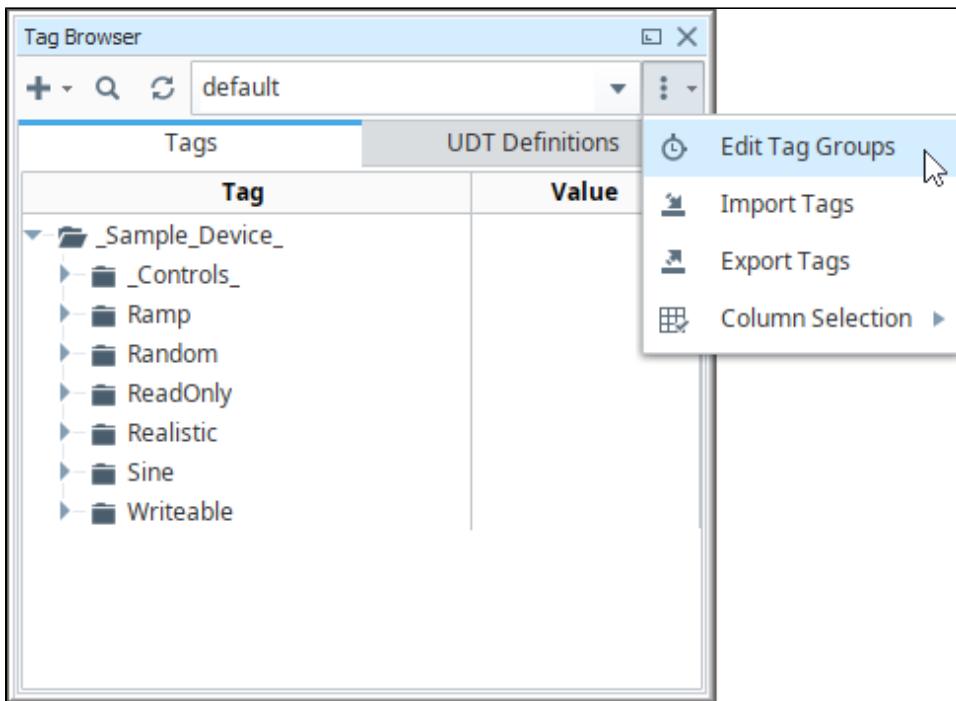
Leased groups are unique in that Tags on the same group may execute at different rates while being in the same group. For example, if Tags **A** and **B** are both on the same Leased group, and all Designer, Vision Client, and Perspective Sessions are closed, then both tags execute using the **Rate** value. If a user launches a Perspective Session and switches to a view where the value of **A** is displayed on a component binding, then **A** will switch to the **Leased/Driven Rate**. However, since **B** isn't displayed anywhere, it will continue to execute using the **Rate** value.



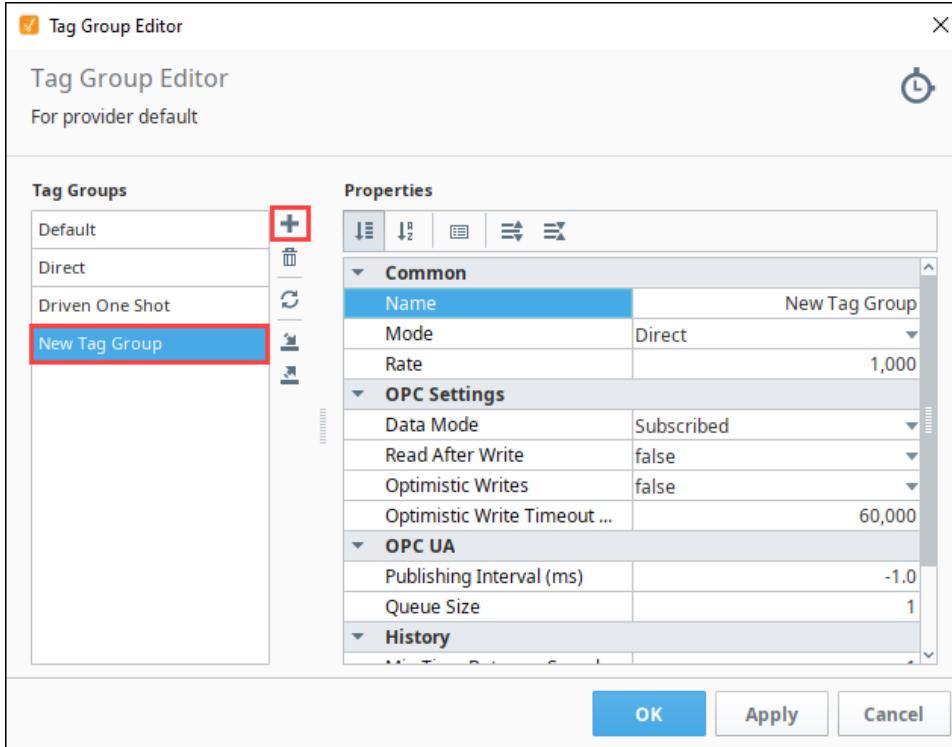
Adding and Editing Tag Groups

Adding and editing Tag Groups is easy in the Designer once you understand how the different Tag Group modes work. It's just a matter of choosing which Tag Group mode you want to use for your Tag, and entering the properties for your Tag Group.

1. In the Tag Browser, click on the **Edit Tag Group** icon under the Options Menu to open the Tag Group Editor window.



2. A list of already configured Tag Groups appear on the left side of the window and configuration settings on the right. To add a Tag Group, click the **Add**  icon. (Alternatively, you can click on an existing Tag Group to edit it.)

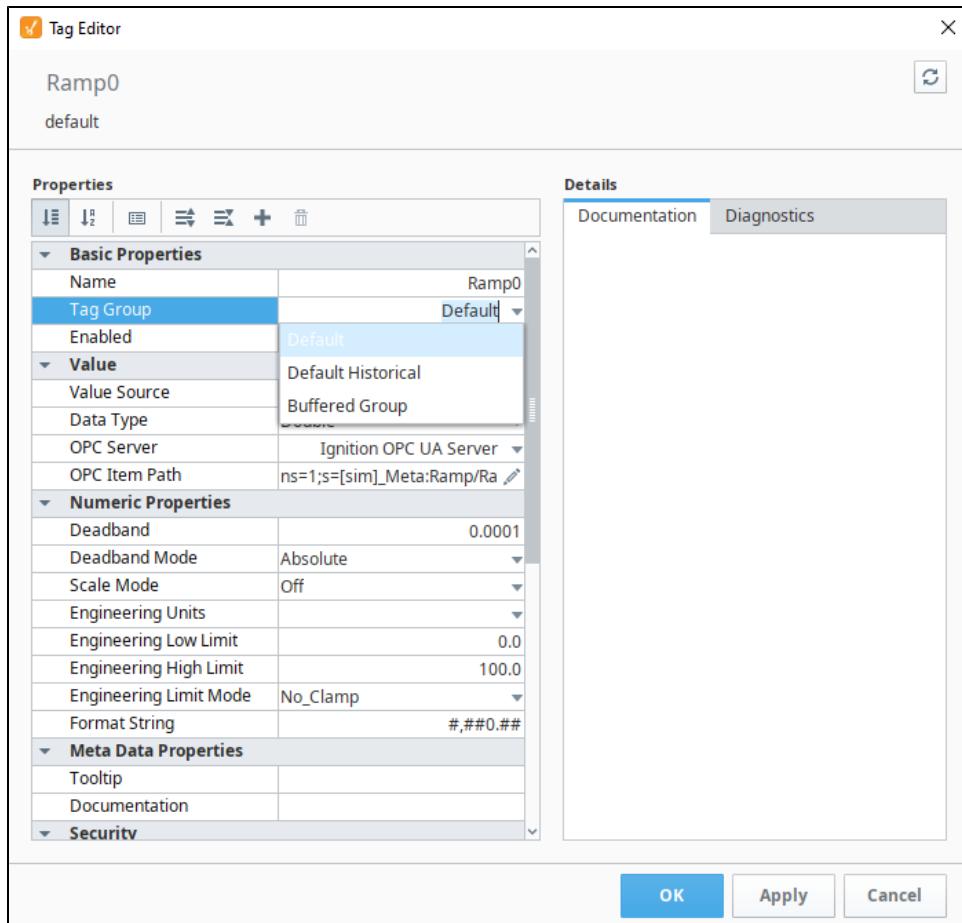


When a Tag Group in the list is selected, the properties for the group will appear on the Properties Table. Each mode will have slightly different settings that will need to be configured. You can find descriptions for those properties in the [Tag Group Properties](#) section below.

Setting a Tag Group on a Tag

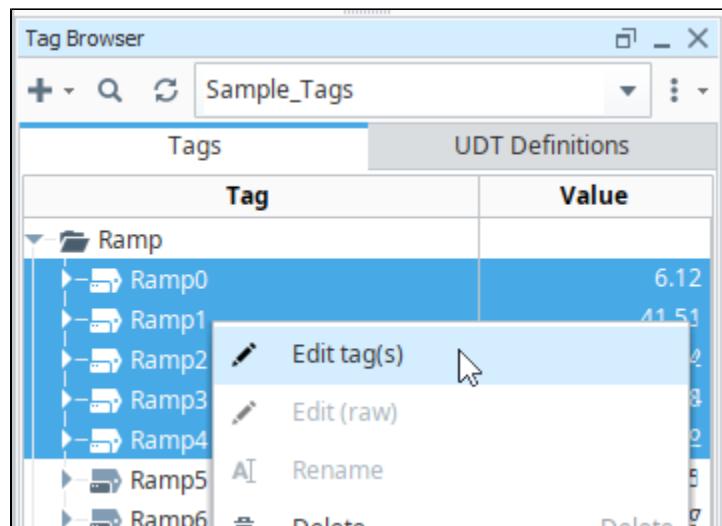
The example below demonstrates how to find the Tag Group on a tag, and change it to something else.

1. In the **Tag Browser**, right-click on any Tag, and click the **Edit tag**  icon. The **Tag Editor** window opens. A list of Tag properties is displayed.
2. Under **Basic Properties**, on the right side of the **Tag Group** property, click the dropdown list and a list of available Tag Groups will appear. Again, Tag Groups are configured per Tag Provider, so this list will only contain Tag Groups that are present in the Tag's Tag Provider.



3. If you wish to change the Tag Group. Select the new option, and click **OK** or **Apply** to apply the new Tag Group. The Tag will switch rates and start using the new Tag Group.

As a reminder, you can select multiple Tags in the Tag Browser by right clicking to edit the Tags. This opens the Tag Editor, and sets the Tag Group for all the selected Tags at the same time.



Tag Group Properties

The following table lists the properties for Tag Groups. Note that some properties are available only for specific modes.

Property	Description	Applicable Modes

Common		
Name	Unique name of the Tag Group.	All
Driven Mode	The rate of the Tag Group is based on the value of a driving Tag. The condition is a simple comparison between a Tag value and a number. If the condition is true, the Tag Group will execute at the fast rate. If false, it will run at the slow rate. There are two exceptions to this: the Any Change operator, and One-shot mode. Using either of these conditions will not run at a rate. Instead, it will be triggered by a change in the driving Tag's value. Keep in mind that the driving Tag can be an Expression Tag that performs complex calculations and references other Tags. In this way, it's possible to create robust Tag Group triggering.	All
Rate	Base update rate, specified in milliseconds, at which Tags will be executed. Note: If the rate is set to 0, the Tag Group will not execute.	All
Leased /Driven Rate	Used by both the Leased and Driven Modes to determine when the Tag Group should run at the fast rate.	Leased, Driven
Driving Expression	The Tag Group executes based on the condition set on the Driving Expression: Tag or Expression.	Driven
Driving Comparison	How the Comparison Value property should be compared to the Driving Tag's value. If the comparison is true, then the Fast Rate will be used by the Tag Group, otherwise, the Slow Rate will be used. The Any Change operator works differently than the other operators: The Tag Group will execute immediately whenever the driving Tag changes value. Using the Any Change operator means that the Tag Group no longer uses the Slow Rate or Fast Rate properties.	Driven
Comparison Value	Used by the Driving Comparison property to determine if the Tag Group should execute at the slow or fast rate.	Driven
One Shot	One-shot will execute once when the comparison condition is true, and not again until the condition becomes false, and subsequently true.	Driven
OPC Settings		
Data Mode	This mode dictates how OPC values are obtained. The default mode, Subscribed , is preferred because it is much more efficient than a read. Subscribed All OPC Tags in the Tag Group will be subscribed according to the Tag Group rate. Values will come in asynchronously as they change. Polled Tags will not be subscribed, but will instead be synchronously read each time the Tag Group executes. This operation is less efficient, but allows more precise control over when values are obtained. This mode is particularly useful when collecting data over a slow or expensive connection for display. When combined with the one-shot execution mode above, and a static Tag tied to a momentary button, it's easy to create a manual refresh button on a screen that pulls data on-demand.	All
Read After Write	When enabled, a read request will be sent immediately after a write request. This means that the value on the Tag will be updated much quicker to reflect the latest written value. Enabling this property is less efficient as a single write to a Tag becomes two separate requests. This is especially helpful with slower Tag Groups as the Tags will show the latest value quicker than the normal execution would allow.	All
Optimistic Writes	Optimistic Writes are only valid on OPC Tags . Optimistic Writes set a newly written Tag value in Ignition before receiving confirmation of the write from the PLC. This helps the operators see their newly entered value right away and is useful if you have slow a Tag Group rate. A faster rate (1 second or quicker) will have less need to turn on Optimistic Writes. If enabled, written values will be applied to the Tag in Ignition immediately. Normally, the system must receive confirmation that a write was successful from the device before the Tag in Ignition's value would change. The Optimistic Writes property changes the behavior by assuming the write went through until the next read value or subscription update proves otherwise. Enabling this will make writes appear to execute much quicker. Works in conjunction with the OPC Optimistic Write Timeout property below. If the Tag in Ignition does not receive confirmation that the new write was successful within the timeout, the Tag will change back to the last known value. While in an ambiguous state, the Tag will have a quality of " Good (Provisional) ". This setting can be paired with the OPC Read After Write : the Ignition Tag will assume the newly written value, while an asynchronous read request is quickly sent out to confirm the write went through.	All

	While the write is pending, values received from subscription activities will override the current value. Assuming an initial value of 0, if a write of 10 is applied to the Ignition Tag, then the Tag will show a value of 10 until the system can confirm the new value. If a subscription update then returns a value of 5, the Ignition Tag will change to 5.	
Optimistic Write Timeout (MS)	The timeout period for Optimistic Writes. A value of 0 effectively disables the fallback functionality: the new value is maintained on the Tag until the next read or subscription activity.	All
OPC UA		
Publishing Interval (ms)	The rate at which data is delivered to the OPC-UA client. A value of -1 means automatic, allowing the OPC-UA client to determine the rate.	All
Queue Size	<p>The OPC-UA specifications states that in cases where the sampling interval (the rate as which the server checks the data source for changes) is faster than the publishing interval (rate at which the data is delivered to the client), the samples may be queued or batched together before publishing. This setting determines the maximum size of that queue. When the maximum is reached and a publish has not yet occurred, oldest samples are dropped first.</p> <p>Currently, there are not many features in Ignition that utilize multiple entries in the queue, but 3rd party OPC-UA clients may be able to take advantage of this setting.</p> <p>Note that values on Ignition Tags will only ever show one value at a time, regardless of what this property is set to. (If the value is 0, nothing will be queued.)</p> <p>Support for this feature is dictated by the driver.</p> <ul style="list-style-type: none"> • DNP3 - See the DNP3 page for details on how buffered data and Sequence of Events works 	All
History		
Min Time Between Samples	Minimum time between samples (integer).	All
Min Time Units	Minimum time in units is defined as: Milliseconds, Seconds, Minutes, Hours, Days, Weeks, Months, and Years.	All
Max Time Between Samples	Maximum time between samples (integer).	All
Max Time Units	Maximum time in units is defined as: Milliseconds, Seconds, Minutes, Hours, Days, Weeks, Months, and Years.	All

Tag Providers

Tag Providers

At the highest level of Tag configuration is the Tag Provider. A provider is a Tag database (a collection of Tags) and a name. An Ignition Gateway can have any number of Tag Providers, and therefore the name is used to distinguish which provider a Tag comes from. Tag Providers can be set up with security or even disabled independent of each other.

Every copy of Ignition has its own Tags, spread across one or more Tag Providers. With the remote Tag Provider, Ignition can also see the Tags on another Gateway, as long as the two Gateways connected through a Gateway network.

All Tags reside in a Tag Provider and have realtime values. Additionally, there is the concept of [Tag historian providers](#), which can store and query historical data for Tags. Each Tag can optionally have a historian provider assigned to it to whom it will report value changes for historical storage.

Realtime Provider Types

There are no differences between the two types of Tag Providers other than where the Tag's configuration is stored and executed. You can have as many Tag providers as you want, and there are security settings to either lock them or open them to the network.

In these examples we refer to two copies of Ignition. The "local" Ignition is the Ignition that you are currently configuring. The "remote" Ignition is another installation of Ignition somewhere on your network.

Internal Tag Provider

Internal Tag Providers store all configuration and do any execution (read, write, history, alarms) through the local Ignition Gateway. Every new Ignition installation automatically creates an Internal Tag Provider named "default." You can add as many Internal Tag Providers as you want.

This provider can be exposed or hidden from other Gateways on the network through the Gateway's OPC UA settings.

Remote Tag Provider

Remote Tag Providers connect a remote installation of Ignition and access those Tags. The remote Tag Provider works by creating a link from the local Gateway to a Tag provider on a remote Gateway using a Gateway Network connection. The local Ignition may be allowed to read and write to the remote Tags, but any execution is handled by the remote Gateway. So, things like writing to a PLC, alarms, and history will still be handled by the remote Ignition.

By default, a Remote Tag Provider will fall under the [Default Security Zone](#) and be read only.

This feature is new in Ignition version **8.1.4**
[Click here](#) to check out the other new features

As of release 8.1.4, remote Tag Providers now support annotations, such as those created by the [Power Chart](#) component or `system.tag.storeAnnotations` function.

Configuring Realtime Providers

Realtime Tags providers are configured in the Gateway's Config section under **Tags > Realtime**. After installation, the Ignition Gateway will start with an internal provider defined. You can edit its name and settings by selecting **edit** to the right of its entry in the table, or create new providers by selecting **Create new Realtime Tag Provider** below the table.

On this page ...

- [Tag Providers](#)
- [Realtime Provider Types](#)
 - [Internal Tag Provider](#)
 - [Remote Tag Provider](#)
- [Configuring Realtime Providers](#)
 - [Standard Tag Provider](#)
 - [Remote Tag Provider](#)



INDUCTIVE
UNIVERSIT

Tag Providers

[Watch the Video](#)



INDUCTIVE
UNIVERSIT

Remote Tag Provider

[Watch the Video](#)

The screenshot shows the Ignition software interface. The top navigation bar includes the Ignition logo, Help, and Get Designer buttons. The left sidebar, titled 'SYSTEM', contains links for Home, Status, Config (which is selected), Overview, Backup/Restore, Licensing, Modules, Projects, Redundancy, Gateway Settings, NETWORKING (with sub-links for Gateway Network and Email Settings), and SECURITY. The main content area is titled 'Realtime Tag Providers'. It displays a table with two rows of providers:

Name	Description	Enabled	Type	
default	Default tag provider	true	Standard Tag Provider	<button>delete</button> <button>edit</button>
Test_Provider		true	Remote Tag Provider (Gateway Network)	<button>delete</button> <button>edit</button>

At the bottom of the table, there is a link to 'Create new Realtime Tag Provider...'. The overall interface has a clean, modern design with a light blue and white color scheme.

There are two types of Realtime Tag Providers that you can choose from:

- Standard Tag Provider
- Remote Tag Provider

Standard Tag Provider

Tags are stored inside of Ignition and executed by the system.

Setting	Description
Name	The name of the provider.
Description	The description of the provider.
Enabled	If true, Tag provider is enabled. Default is true.
Default Database	The default database connection to use for expression Tags that run SQL queries. All query Tags with default database providers selected will run their queries against this database source.
Tag Editing Permissions	<p>Determines the roles required to edit, create, or delete Tags in the provider. Expects a path to a Security Level. Multiple levels can be specified, separated by a comma. For example:</p> <pre>Authenticated/Roles/Administrator, SecurityZones/MyZone</pre> <p>When multiple security levels are provided, the radio buttons determine if the user needs all of the listed security levels, or at least one.</p> <div style="background-color: #ffd700; padding: 5px;"> <p>This feature is new in Ignition version 8.1.2 Click here to check out the other new features</p> </div> <p>As of version 8.1.2, Edge Gateways now have access to the Tag Editing Permissions setting.</p>
Advanced Properties	
Allow Back-fill Data	<div style="background-color: #ffd700; padding: 5px;"> <p>This feature is new in Ignition version 8.1.4 Click here to check out the other new features</p> </div> <p>If enabled, data will be allowed to arrive out of order from the source. Data from the past will be stored to history, but will not be used for alarms, scripts, or subscriptions. If false (default behavior), each value will be processed fully as it arrives. Default is false.</p>

Remote Tag Provider

Tag Provider from one Gateway is brought in to another Gateway.

Setting	Description
Name	The name of the provider.
Description	The description of the provider
Gateway	The name of the Gateway on the Gateway Network that this provider is coming from.
Provider	The name of the provider as it is on the remote Gateway. This does not have to be the same as its name on the new Gateway.
History Access Mode	This setting dictates how Tag history is queried for remote Tags. Gateway Network will go through the Gateway Network to query history. Database will query the database directly.
History Datasource	The datasource to query when History Access Mode is set to Database.
History Driver	If querying the database directly, this is the Gateway name of the remote system. It is used to identify data from that system in the database.
History Provider	If querying the database directly, this is the name of the Tag provider on the remote system. It is used along with driver name to identify the correct Tags in the database.
Alarms Enabled	If true, alarms configured on the remote Gateway will be enabled on the new Gateway.
Alarm Mode	How alarm state should be monitored. In 'queried', state will be queried through the Gateway network when necessary. In 'subscribed', the state will be subscribed, and updates will be sent asynchronously. Subscribed provides better performance, but uses more memory.
Advanced Properties	
Allow Back-fill Data	<p>This feature is new in Ignition version 8.1.4. Click here to check out the other new features</p> <p>If enabled, data will be allowed to arrive out of order from the source. Data from the past will be stored to history, but will not be used for alarms, scripts, or subscriptions. If false (default behavior), each value will be processed fully as it arrives. Default is false.</p>

Related Topics ...

- [User Defined Types - UDTs](#)
- [Tag Groups](#)
- [Tag Browser](#)

Tag Event Scripts

Scripts can be attached to Tags. When you edit a Tag, you can navigate to the Tag Events section and click on the **Edit**  icon to see a list of all of the available events. Those events are

- Value Changed
- Quality Changed
- Alarm Active
- Alarm Cleared
- Alarm Acknowledged

Each event is unique and can have specialized arguments depending on the event. Also, because Tags are stored in the Gateway, all these scripts fire in the [Gateway scope](#).

Note: Because these scripts are Gateway scoped, certain things like print statements will not print to the Designer console, but will print instead to the wrapper log file in Ignition's installation directory. Keep that in mind when doing any testing.

On this page ...

- Event Options
 - Value Changed
 - Quality Changed
 - Alarm Active
 - Alarm Cleared
 - Alarm Acknowledged
- Using the Project Library in a Tag Event Script
- UDT Parameters in Tag Event Scripts
 - Modern Approach
 - Legacy Approach
- Tag Script Examples



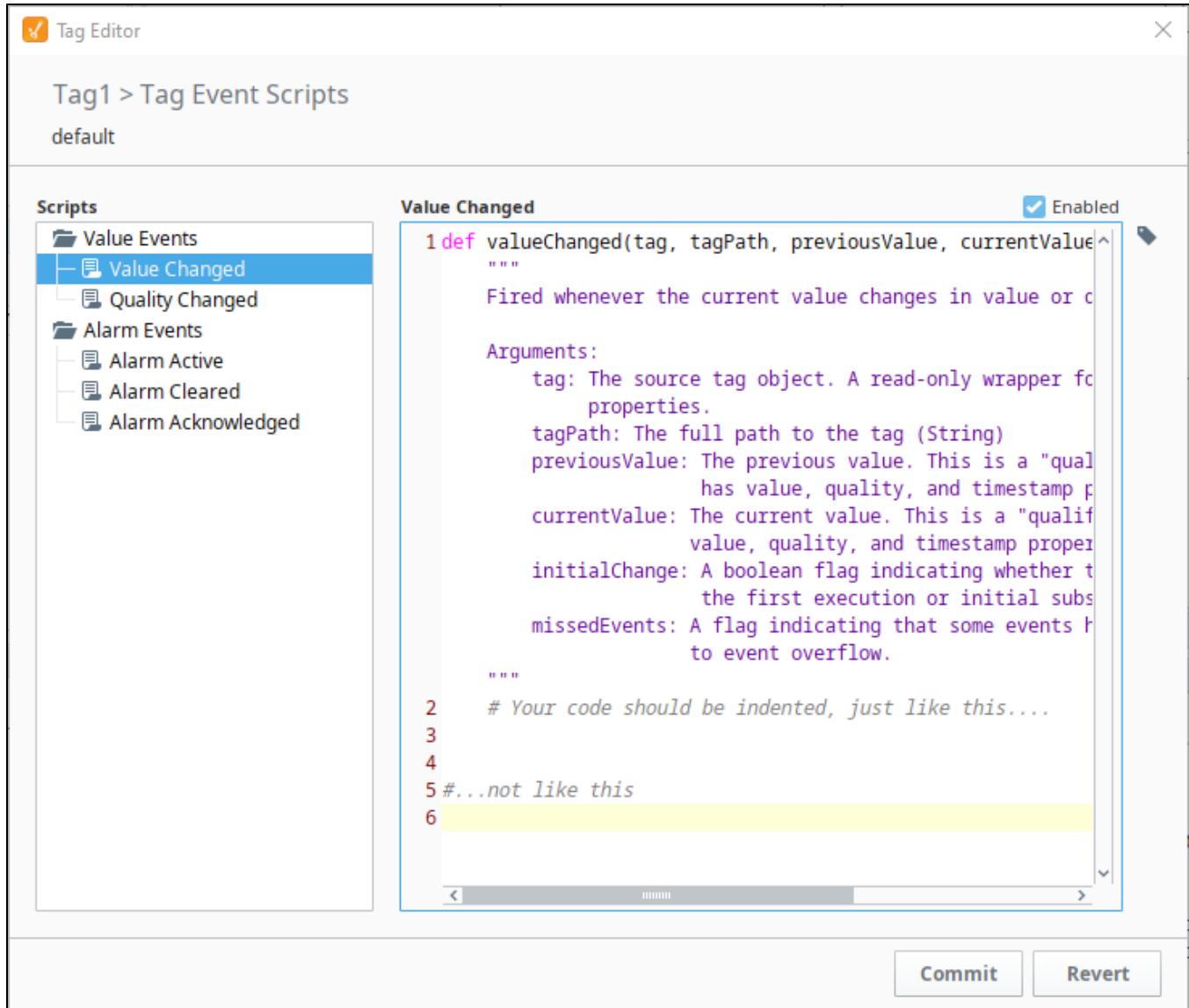
INDUCTIVE
UNIVERSITY

Tag Scripts

[Watch the Video](#)

Event Options

Once an event has been selected, note that the top of the text area is [defining a function](#). As a result, all code that should execute when this event occurs should be indented at least once to be included in the definition.



Value Changed

The Value Changed event is fired whenever the value of this particular Tag changes. This event has a variety of arguments available for use in the script:

- String tagPath - The full path to the Tag. Example: [tagProvider]Folder/Folder/Tag
- Object previousValue - The previous value. This is a qualified value, so it has value, quality, and timestamp properties.
- Object currentValue - The current value. This is a qualified value, so it has value, quality, and timestamp properties.
- Boolean initialChange - A boolean flag indicating whether this event is due to the initial subscription or the first execution after a Tag update.
- Boolean missedEvents - A flag indicating that some events have been skipped due to an event overflow.



The currentValue and previousValue arguments are qualified values: objects that contain a value, timestamp, and quality. This means that to get to the value of the currentValue, your script would need to access currentValue.value

Quality Changed

The Quality Changed event is fired whenever the quality of this particular Tag changes. Since Tags use a QualifiedValue, which include a value, quality, and timestamp, this script will fire whenever any of those change. Refer to [Scripting Object Reference](#) for more information. This event has a variety of arguments available for use in the script:

- String tagPath - The full path to the Tag. Example: [tagProvider]Folder/Folder/Tag

- Object previousValue - The previous value. This is a qualified value, so it has value, quality, and timestamp properties.
- Object currentValue - The current value. This is a qualified value, so it has value, quality, and timestamp properties.
- Boolean initialChange - A boolean flag indicating whether this event is due to the initial subscription or the first execution after a Tag update.
- Boolean missedEvents - A flag indicating that some events have been skipped due to an event overflow.

Alarm Active

The Alarm Active event fires whenever a new active alarm event occurs. This event has a variety of arguments available for use in the script:

- String tagPath - The full path to the Tag. Example: [tagProvider]Folder/Folder/Tag
- String alarmName - The name of the alarm. This does not include the full alarm path.
- Object alarmEvent - The full alarm event object. The properties available to this object are:
 - eventId
 - source
 - name
 - priority
 - displayPath
 - displayPathOrSource (the display path if it is set, otherwise the source)
 - state (the current state, active/cleared + acked/unacked)
 - eventState (the last transition, active, clear, acknowledged)
 - isClear (a boolean if the alarm is currently cleared)
 - isAcked (a boolean if the alarm is currently acknowledged)
 - isShelved (a boolean if the alarm is currently shelved)
 - notes
- String alarmPath - The full alarm path.
- Boolean missedEvents - A flag indicating that some events have been skipped due to an event overflow.

Alarm Cleared

The Alarm Cleared event fires whenever an alarm becomes cleared on the Tag. This event has a variety of arguments available for use in the script:

- String tagPath - The full path to the Tag. Example: [tagProvider]Folder/Folder/Tag
- String alarmName - The name of the alarm. This does not include the full alarm path.
- Object alarmEvent - The full alarm event object. See the Alarm Active alarmEvent object for the full list of available properties.
- String alarmPath - The full alarm path.
- Boolean missedEvents - A flag indicating that some events have been skipped due to an event overflow.

Alarm Acknowledged

The Alarm Acknowledged event fires whenever an alarm event becomes acknowledged on the Tag. This event has a variety of arguments available for use in the script:

- String tagPath - The full path to the Tag. Example: [tagProvider]Folder/Folder/Tag
- String alarmName - The name of the alarm. This does not include the full alarm path.
- Object alarmEvent - The full alarm event object. See the Alarm Active alarmEvent object for the full list of available properties.
- String alarmPath - The full alarm path.
- String ackedBy - The full path to the user that acknowledged the alarm.
- Boolean missedEvents - A flag indicating that some events have been skipped due to an event overflow.

Using the Project Library in a Tag Event Script

Scripts defined in a project script can be called from a Tag Event Script. However, only scripts defined in the Gateway Script Project may be used. For more information on configuring the Gateway Scripting Project, please see the [Project Library](#) page.

UDT Parameters in Tag Event Scripts

Parameters on UDTs can be interacted with from Tag Event Scripts. There are two main approaches.

Modern Approach

Parameters values can be accessed as dictionary values. The benefit of this approach is that value changes to UDT parameters will be reflected in subsequent calls. In most cases, this modern approach is preferable. Thus, if the script needs to access the most recent parameter values on a UDT, and the parameters can change through a means other than editing the UDT (which would restart the tag), then this approach should be used.

If trying to access the value of a parameter named "myParam" from a Tag Event Script within the UDT, the script would look like:

```
# Reminder: "tag" is a built-in argument on all Tag Event Scripts. Accessing the "parameters" key on the tag
# argument will
# provide access to all UDT parameters.
paramValue = tag['parameters']['myParam']
```

Legacy Approach

In Ignition release 7.9 and prior, UDT parameters could be accessed in Tag Event Scripts with the familiar curly brackets approach. Thus, if trying to access the value of a parameter named "myParam" from a Tag Event Script within the UDT, the script would look like:

```
paramValue = {myParam}
```

A large caveat with this approach is that value changes made to the parameter ("myParam", in the example above) would not be reflected in scripts until the UDT was restarted. UDT parameters are pre-compiled, which in this case means value changes are mostly ignored until the UDT is restarted.

In all cases, the modern approach above is preferred.

Tag Script Examples

Printing all parameters

```
# This script will fetch all of the possible parameters in the Tag Changed Script.  
# Note that this will not print out to the console, but it will print to the wrapper logs located on the  
Gateway.  
  
path = tagPath  
prev = previousValue  
cur = currentValue  
init = initialChange  
missed = missedEvents  
print path, prev.value, cur.quality, init, missed
```

Automatic Reset

```
# This code can be used on a Value Changed script, and will automatically reset the value of the tag to 0  
after it goes above 3000.  
# This can be useful for counter tags.  
if currentValue.value > 3000:  
    system.tag.writeBlocking(["[default]IntegerCounterTag"], [0])
```

Copy to another Tag

```
# This code can be used on a Value Changed script, and will record the highest value of the current tag onto  
another memory tag.  
# This can be useful for recording the highest temperature.  
highestRecordedTemp = system.tag.read(["[default]HighestTempTag"]).value  
if currentValue.value > highestRecordedTemp:  
    system.tag.writeBlocking(["[default]HighestTempTag"], [currentValue.value])
```

Automatically reacting to an alarm

```
# This code can be used on an Alarm Active script, and can be used to automatically react to an alarm to  
prevent a disaster from occurring.  
# if personnel were not able to react in time.  
if alarmName == "Tank Pressure Critical":  
    system.tag.writeBlocking(["[default]PressureReleaseValveTag"], [1])  
if alarmName == "Tank Pressure Too Low":  
    system.tag.writeBlocking(["[default]TankFillTag"], [1])
```


Tag Properties

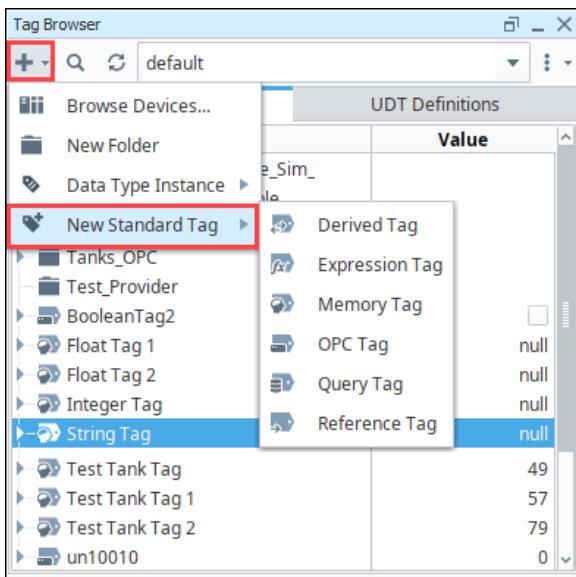
Tags are points of data and may have static values or dynamic values that come from an OPC address, an Expression, or a SQL query. The values can be used on screens, in Transaction Groups, and more. Additionally, Tags offer a core set of features above and beyond simple values, such as scaling, alarming, and history logging. Depending on the specific [type of Tag](#), even more options are available. In general, Tags provide a common interface for tying together many types of data in Ignition.

Tag Configuration in the Designer

Tags are managed in the Tag Editor. To configure a Tag, right-click on it and select **Edit Tag**. Or create a new Tag by right-clicking on the Tags folder in the Tag Browser and use the **New Tag** option to select a new Tag type.

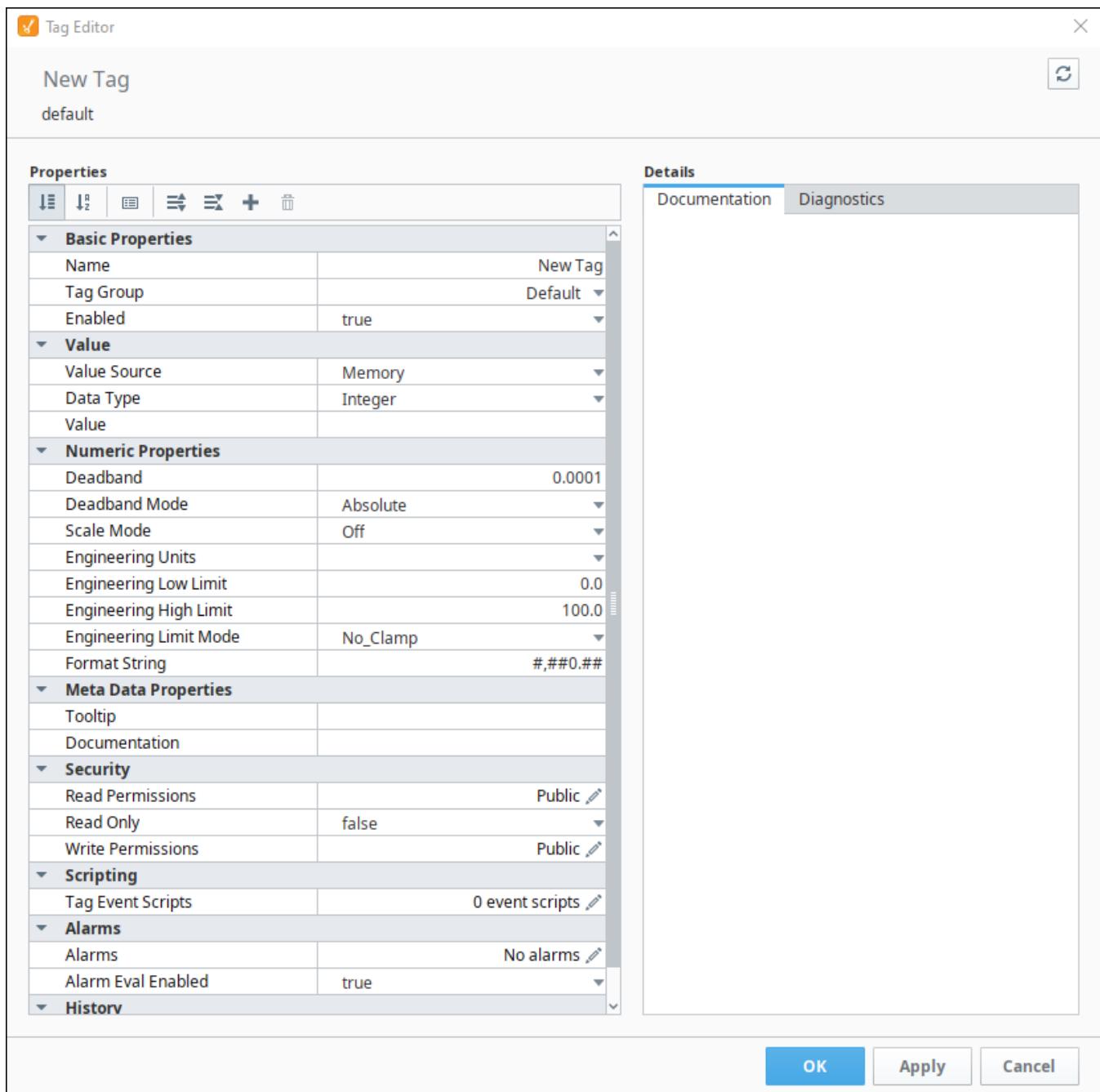
On this page ...

- [Tag Configuration in the Designer](#)
- [Tag Object Types](#)
- [Standard Tag Properties Table](#)
- [Custom Tag Properties](#)
- [Vision Client Tags](#)



Once the **Tag Editor** window is displayed you can set the properties for that Tag. The **Tag Editor** window has the following sections depending on the type of Tag you are creating:

- Basic Properties
- Value
- Numeric Properties
- Meta Data Properties
- Security
- Scripting
- Alarms
- History



Tag Object Types

Some features, such as `system.tag.browse`, can access the Object Type of the tag (sometimes called "tagType"). Below is a table representing the possible types.

Tag Object	Type
Property	A single value underneath an node.
Node	An entity that may have a value and may have children. "Node" is a generic term for other objects in this table, such as a Folder or AtomicTag.
Folder	Represented by a folder in the Tag Browser. Folders generally have child nodes, but don't have values or other properties that make up a tag.
AtomicTag	A "normal" type of tag. Objects with this type can be one of the following (based on the Value Source property):

	<ul style="list-style-type: none"> • OPC Tag • Query Tag • Expression Tag • Derived Tag • Reference Tag • Memory Tag
UdtInstance	<p>An instance of a complex tag instance (UDT Instance). It's important to note that UdtInstances contain other nodes, so this type is generally only seen at the root of a UDT instance.</p> <p>Thus, nodes under a UdtInstance are not considered to have a type of "UdtInstance", unless the child node is actually a UdtInstance: in other words, a nested UDT instance.</p>
UdtType	Represents the root of a complex tag definition (UDT Definition). Similar to UdtInstance, nodes under a UdtType have their own object type, so a UdtType represents the root of a complex tag.
Provider	Represents a Tag Provider.

Standard Tag Properties Table

This following table provides detail on each Tag Property, including the binding name, description, data type, and the Tag Types that use the property.

Property	JSON /Scripting Name	Description	Data Type																					
Basic Properties																								
Name	name	How the Tag will be presented and referenced in the system. The Tag path will be the provider, the folder structure, and this name.	String																					
Tag Group	tagGroup	The Tag Group that will execute the Tag. The Tag Group dictates the rate and conditions on which the Tag will be evaluated. For more details, see Tag Groups .	String																					
Enabled	enabled	Whether the Tag will be evaluated by the Tag Group. If false, the Tag will still be present, but will have a bad quality.	Boolean																					
Value																								
Value Source	valueSource	<p>Specifies how the Tag determines its value. In other words, sets the type of the Tag (Memory, OPC, Expression, etc).</p> <table border="1"> <tr> <th>Value Source</th><th>JSON Name</th></tr> <tr> <td>Derived</td><td>derived</td></tr> <tr> <td>Expression</td><td>expr</td></tr> <tr> <td>Memory</td><td>memory</td></tr> <tr> <td>OPC</td><td>opc</td></tr> <tr> <td>Query</td><td>db</td></tr> <tr> <td>Reference</td><td>reference</td></tr> </table>	Value Source	JSON Name	Derived	derived	Expression	expr	Memory	memory	OPC	opc	Query	db	Reference	reference	String							
Value Source	JSON Name																							
Derived	derived																							
Expression	expr																							
Memory	memory																							
OPC	opc																							
Query	db																							
Reference	reference																							
Data Type	dataType	<p>The data type of the Tag. It is important that this be set as correctly as possible with regards to the Tag's underlying data source. The Tag system will attempt to coerce any raw incoming value (for example, from OPC or a SQL query) into the desired type. For detailed information, see Tag Data Types.</p> <table border="1"> <tr> <th>Data Type</th><th>String Value</th><th>Integer Value</th></tr> <tr> <td>Byte</td><td>Int1</td><td>0</td></tr> <tr> <td>Short</td><td>Int2</td><td>1</td></tr> <tr> <td>Integer</td><td>Int4</td><td>2</td></tr> <tr> <td>Long</td><td>Int8</td><td>3</td></tr> <tr> <td>Float</td><td>Float4</td><td>4</td></tr> <tr> <td>Double</td><td>Float8</td><td>5</td></tr> </table>	Data Type	String Value	Integer Value	Byte	Int1	0	Short	Int2	1	Integer	Int4	2	Long	Int8	3	Float	Float4	4	Double	Float8	5	String
Data Type	String Value	Integer Value																						
Byte	Int1	0																						
Short	Int2	1																						
Integer	Int4	2																						
Long	Int8	3																						
Float	Float4	4																						
Double	Float8	5																						

		<table border="1"> <tr><td>Boolean</td><td>Boolean</td><td>6</td></tr> <tr><td>String</td><td>String</td><td>7</td></tr> <tr><td>DateTime</td><td>DateTime</td><td>8</td></tr> <tr><td>Text</td><td>Text</td><td>10</td></tr> <tr><td>Byte Array</td><td>Int1Array</td><td>17</td></tr> <tr><td>Short Array</td><td>Int2Array</td><td>18</td></tr> <tr><td>Integer Array</td><td>Int4Array</td><td>11</td></tr> <tr><td>Long Array</td><td>Int8Array</td><td>12</td></tr> <tr><td>Float Array</td><td>Float4Array</td><td>19</td></tr> <tr><td>Double Array</td><td>Float8Array</td><td>13</td></tr> <tr><td>Boolean Array</td><td>BooleanArray</td><td>14</td></tr> <tr><td>String Array</td><td>StringArray</td><td>15</td></tr> <tr><td>DateTime Array</td><td>DateTimeArray</td><td>16</td></tr> <tr><td>Binary Data</td><td>ByteArray</td><td>20</td></tr> <tr><td>Dataset</td><td>DataSet</td><td>9</td></tr> <tr><td>Document</td><td>Document</td><td>29</td></tr> </table>	Boolean	Boolean	6	String	String	7	DateTime	DateTime	8	Text	Text	10	Byte Array	Int1Array	17	Short Array	Int2Array	18	Integer Array	Int4Array	11	Long Array	Int8Array	12	Float Array	Float4Array	19	Double Array	Float8Array	13	Boolean Array	BooleanArray	14	String Array	StringArray	15	DateTime Array	DateTimeArray	16	Binary Data	ByteArray	20	Dataset	DataSet	9	Document	Document	29	
Boolean	Boolean	6																																																	
String	String	7																																																	
DateTime	DateTime	8																																																	
Text	Text	10																																																	
Byte Array	Int1Array	17																																																	
Short Array	Int2Array	18																																																	
Integer Array	Int4Array	11																																																	
Long Array	Int8Array	12																																																	
Float Array	Float4Array	19																																																	
Double Array	Float8Array	13																																																	
Boolean Array	BooleanArray	14																																																	
String Array	StringArray	15																																																	
DateTime Array	DateTimeArray	16																																																	
Binary Data	ByteArray	20																																																	
Dataset	DataSet	9																																																	
Document	Document	29																																																	
		<p>Note: Regarding Array data types, Alarming, Scaling, and Historical settings applied to an array Tag are propagated down to elements in the array.</p>																																																	
Value	value	The value of the Tag. Can only be modified if the Tag allows value writing and the user has sufficient privileges.	Object (depends on the dat atype of the Tag)																																																
OPC Server	opcServer	Only present for OPC Tags. The server against which to subscribe the data point. Only present for OPC Tags.																																																	
OPC Item Path	opcItemPath	Only present for OPC Tags. The path to subscribe to on the server. The point will be subscribed at the rate dictated by the Tag Group.																																																	
Source Tag Path	sourceTagPath	The path to the Tag that this Tag is referencing. Only present for Reference and Derived Tags.																																																	
Execution Mode	executionMode	<p>Only present for Query and Expression Tags . Determines how when the Tag executes.</p> <ul style="list-style-type: none"> • Event Driven - Updates when something happens (i.e., value event or alarm event) within the expression. • Fixed Rate - Tag will be executed at the set or fixed rate. Adds the Execution Rate property, which determines how often the Tag executes in milliseconds. • Tag Group - Tags are executed by Tag Groups, which dictate the rate of execution. <table border="1"> <thead> <tr> <th>Execution Mode</th><th>JSON Name</th></tr> </thead> <tbody> <tr><td>Event Driven</td><td>EventDriven</td></tr> <tr><td>Fixed Rate</td><td>FixedRate</td></tr> <tr><td>Tag Group</td><td>TagGroupRate</td></tr> </tbody> </table>		Execution Mode	JSON Name	Event Driven	EventDriven	Fixed Rate	FixedRate	Tag Group	TagGroupRate																																								
Execution Mode	JSON Name																																																		
Event Driven	EventDriven																																																		
Fixed Rate	FixedRate																																																		
Tag Group	TagGroupRate																																																		
Expression	expression	The expression the Tag will use to determine its value.	String																																																
Read Expression	deriveExpressionGetter	The expression that determines how the value on the Derived Tag is displayed.	String																																																
Query	query	The SQL query to be run, which drives the tag's value. Queries doing database reads and writes are possible, see the Query Type property description for details.																																																	
Write Expression	deriveExpressionSetter	The expression that determines how the value on the Derived Tag is displayed.	String																																																
Datasource	datasource	The database connection that the query Tag will execute against.	String																																																
Query Type	queryType		String																																																

This feature is new in Ignition version **8.1.3**
[Click here](#) to check out the other new features

Defines whether the query is executing a database read or a database write. Important for determining the **value** behavior of the Tag.

Possible values are:

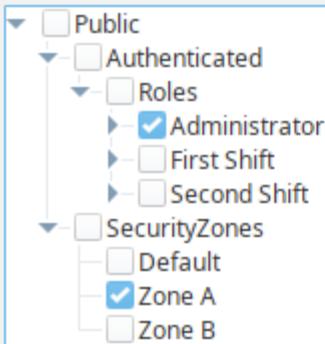
- **AutoDetect** - Query type is determined from the query itself.
- **Select** - Dictates that the query is reading data from the database. The query result set will be stored on the Tag's value.
- **Update** - Dictates that the query is writing data to the database (but does not require an UPDATE statement in the query, specifically). The value on the query Tag will be the number of affected rows.

Numeric Properties

Deadband	deadband	A numerical value used to prevent unnecessary updates for Tags whose values change by small amounts.	Numeric																		
Deadband Mode	deadbandMode	<p>Defines how the deadband value is used.</p> <ul style="list-style-type: none"> • Absolute - The deadband setting is considered to be an absolute value. • Percent - The actual deadband is calculated as a percent of the Tag's engineering unit span. <p>Valid values are as follows:</p> <table border="1"> <thead> <tr> <th>Deadband Mode</th><th>JSON Name</th></tr> </thead> <tbody> <tr> <td>Absolute</td><td>Absolute</td></tr> <tr> <td>Percent</td><td>Percent</td></tr> </tbody> </table>	Deadband Mode	JSON Name	Absolute	Absolute	Percent	Percent	String												
Deadband Mode	JSON Name																				
Absolute	Absolute																				
Percent	Percent																				
Scale Mode	scaleMode	<p>If and how the Tag value will be scaled between the source, and what is reported for the Tag.</p> <p>Valid values are as follows:</p> <table border="1"> <thead> <tr> <th>Mode</th><th>JSON Name</th><th>Int Value</th></tr> </thead> <tbody> <tr> <td>Off</td><td>Off</td><td>0</td></tr> <tr> <td>Linear</td><td>Linear</td><td>1</td></tr> <tr> <td>Square Root</td><td>SquareRoot</td><td>2</td></tr> <tr> <td>Exponential Filter</td><td>ExponentialFilter</td><td>3</td></tr> <tr> <td>Bit Inversion</td><td>BitInversion</td><td>4</td></tr> </tbody> </table>	Mode	JSON Name	Int Value	Off	Off	0	Linear	Linear	1	Square Root	SquareRoot	2	Exponential Filter	ExponentialFilter	3	Bit Inversion	BitInversion	4	String
Mode	JSON Name	Int Value																			
Off	Off	0																			
Linear	Linear	1																			
Square Root	SquareRoot	2																			
Exponential Filter	ExponentialFilter	3																			
Bit Inversion	BitInversion	4																			
Raw Low	rawLow	Start of the "raw" value range. Only present if Scale Mode is set to Linear or Square Root .	Numeric																		
Raw High	rawHigh	End of the "raw" value range. Only present if Scale Mode is set to Linear or Square Root .	Numeric																		
Scaled Low	scaledLow	Start of "scaled" value range. Raw low will map to Scaled low for the Tag. Only present if Scale Mode is set to Linear or Square Root .	Numeric																		
Scaled High	scaledHigh	End of "scaled" value range. Raw high will map to Scaled high for the Tag. Only present if Scale Mode is set to Linear or Square Root	Numeric																		
Clamp Mode	clampMode	<p>How values that fall outside of the ranges will be treated. "Clamped" values will be adjusted to the low/high scaled value as appropriate. Only present if Scale Mode is set to Linear or Square Root.</p> <p>Valid values are as follows:</p> <table border="1"> <thead> <tr> <th>Clamp Mode</th><th>JSON Name</th><th>Int Value</th></tr> </thead> <tbody> <tr> <td>No_Clamp</td><td>No_Clamp</td><td>0</td></tr> <tr> <td>Clamp_Low</td><td>Clamp_Low</td><td>1</td></tr> <tr> <td>Clamp_High</td><td>Clamp_High</td><td>2</td></tr> <tr> <td>Clamp_Both</td><td>Clamp_Both</td><td>3</td></tr> </tbody> </table>	Clamp Mode	JSON Name	Int Value	No_Clamp	No_Clamp	0	Clamp_Low	Clamp_Low	1	Clamp_High	Clamp_High	2	Clamp_Both	Clamp_Both	3	String			
Clamp Mode	JSON Name	Int Value																			
No_Clamp	No_Clamp	0																			
Clamp_Low	Clamp_Low	1																			
Clamp_High	Clamp_High	2																			
Clamp_Both	Clamp_Both	3																			
Scale Factor	scaleFactor	For single parameter modes (currently only Exponential Filter), the factor parameter for the equation. Used when the Scale Mode property is set to Exponential Filter	Numeric																		

Engineering Units	engUnit	The engineering units of the value.	String															
Engineering Low Limit	engLow	The lowest expected value of the Tag.	Numeric															
Engineering High Limit	engHigh	The highest expected value of the Tag.	Numeric															
Engineering Limit Mode	engLimitMode	<p>Dictates how the engineering range should be enforced on the Tag. If not "Off", the Tag will change to bad quality ("limit exceeded"), when the value falls outside the specified range.</p> <p>Valid values are as follows:</p> <table border="1"> <thead> <tr> <th>Limit Enforcement</th> <th>JSON Name</th> <th>Int Value</th> </tr> </thead> <tbody> <tr> <td>No_Clamp</td> <td>No_Clamp</td> <td>0</td> </tr> <tr> <td>Clamp_Low</td> <td>Clamp_Low</td> <td>1</td> </tr> <tr> <td>Clamp_High</td> <td>Clamp_High</td> <td>2</td> </tr> <tr> <td>Clamp_Both</td> <td>Clamp_Both</td> <td>3</td> </tr> </tbody> </table>	Limit Enforcement	JSON Name	Int Value	No_Clamp	No_Clamp	0	Clamp_Low	Clamp_Low	1	Clamp_High	Clamp_High	2	Clamp_Both	Clamp_Both	3	Numeric
Limit Enforcement	JSON Name	Int Value																
No_Clamp	No_Clamp	0																
Clamp_Low	Clamp_Low	1																
Clamp_High	Clamp_High	2																
Clamp_Both	Clamp_Both	3																
Format String	formatString	<p>How the value should be formatted when converted to a string (only applies to numerical data types). Uses # and 0 characters to describe the format.</p> <p># : If the number in this position is non-zero, then do not show the position. Otherwise, show the number. Useful when you only want to show a decimal position if the value is non-zero.</p> <p>0 : If the number in this position is non-zero, then show that number. Otherwise, show a zero. Useful to add leading and trailing zeros to a value.</p> <p>See Data Type Formatting Reference.</p>	String															
Meta Data Properties																		
Tooltip	tooltip	The tooltip provides a hint to visual components as to what should be displayed when the user hovers their mouse cursor over the component that is being driven by the value of this Tag.	String															
Documentation	documentation	A freeform text property for information about the Tag.	String															
Security																		
Read Permissions	readPermissions	<p>Defines the security levels required in order to read values from a Tag. For more information, see Tag Security Properties. Contains the following elements:</p> <table border="1"> <thead> <tr> <th>Name</th> <th>JSON Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Type</td> <td>type</td> <td>Represents the selected radio button on the security level UI, determining if all of the elements in the securityLevels array are required, or if any of the elements are allowed. Possible values are: AnyOf AllOf</td> </tr> <tr> <td>Security Levels</td> <td>securityLevels</td> <td>Represents allowed security levels for this permission. Each level is represented as a JSON object, containing a "name" value that represents the name of a security level, and a "children" array which represents any levels under the current. The actual "selected" levels are any levels that have an empty "children" object. See the example below for more information.</td> </tr> </tbody> </table> <p>The JSON in this example uses the configuration shown in the image below. Permission is granted if the security levels on the request are from either an "Administrator" user, or if the request originated from the "Zone A" Security Zone.</p>	Name	JSON Name	Description	Type	type	Represents the selected radio button on the security level UI, determining if all of the elements in the securityLevels array are required, or if any of the elements are allowed. Possible values are: AnyOf AllOf	Security Levels	securityLevels	Represents allowed security levels for this permission. Each level is represented as a JSON object, containing a "name" value that represents the name of a security level, and a "children" array which represents any levels under the current. The actual "selected" levels are any levels that have an empty "children" object. See the example below for more information.	JSON Object						
Name	JSON Name	Description																
Type	type	Represents the selected radio button on the security level UI, determining if all of the elements in the securityLevels array are required, or if any of the elements are allowed. Possible values are: AnyOf AllOf																
Security Levels	securityLevels	Represents allowed security levels for this permission. Each level is represented as a JSON object, containing a "name" value that represents the name of a security level, and a "children" array which represents any levels under the current. The actual "selected" levels are any levels that have an empty "children" object. See the example below for more information.																

Security Levels (including Roles) may be added by going to the [Config > Security > Security Levels](#) page of the Gateway Web Interface



- The security levels of the user must match **all** of the required security levels
- At least one of the security levels of the user must match **any** of the required security levels

```

"readPermissions": {
    "type": "AnyOf",
    "securityLevels": [
        {
            "name": "Authenticated",
            "children": [
                {
                    "name": "Roles",
                    "children": [
                        {
                            "name": "Administrator",
                            "children": []
                        }
                    ]
                }
            ]
        },
        {
            "name": "SecurityZones",
            "children": [
                {
                    "name": "Zone A",
                    "children": []
                }
            ]
        }
    ]
}
  
```

Read Only	readOnly	Defines whether a Tag is read-only or writeable. For more information, see Tag Security Properties .	value: boolean
Write Permissions	writePermissions	Defines the security levels required in order to read values from a Tag. For more information, see Tag Security Properties . Contains the following elements:	JSON Object

Name	JSON Name	Description
Type	type	

		Represents the selected radio button on the security level UI, determining if all of the elements in the securityLevels array are required, or if any of the elements are allowed. Possible values are: AnyOf AllOf
Security Levels	securityLevels	Represents allowed security levels for this permission. Each level is represented as a JSON object, containing a "name" value that represents the name of a security level, and a "children" array which represents any levels under the current. The actual "selected" levels are any levels that have an empty "children" object. See the example below for more information.

The JSON in this example uses the configuration shown in the image below. Permission is granted if the security levels on the request are from either an "Administrator" user, or if the request originated from the "Zone A" Security Zone.

💡 Security Levels (including Roles) may be added by going to the [Config > Security > Security Levels](#) page of the Gateway Web Interface

The security levels of the user must match **all of** the required security levels
 At least one of the security levels of the user must match **any of** the required security levels

```
"writePermissions": {
    "type": "AnyOf",
    "securityLevels": [
        {
            "name": "Authenticated",
            "children": [
                {
                    "name": "Roles",
                    "children": [
                        {
                            "name": "Administrator",
                            "children": []
                        }
                    ]
                }
            ]
        },
        {
            "name": "SecurityZones",
            "children": [
                {
                    "name": "Zone A",
                    "children": []
                }
            ]
        }
    ]
}
```

Scripting																					
Tag Event Scripts	eventScripts	<p>Each Tag has the option to have Tag Event Scripts on it. When you edit a Tag, you can navigate to the Tag Events screen to see a list of all of the Tag scripts. You can then select which event you would like to write a script for. You can even write a script for multiple events if you like. For detailed information, see Tag Event Scripts.</p> <p>When interacting with a Tag from a script, the Tag Event Scripts are represented as an array of JSON objects. Each JSON object is described in the expandable area below:</p> <p>Key Description</p> <table border="1"> <thead> <tr> <th>Key</th><th>Description</th></tr> </thead> <tbody> <tr> <td>eventid</td><td>A value representing the type of event script</td></tr> <tr> <td>script</td><td>A value representing the content of the script</td></tr> </tbody> </table> <p>Possible eventid values</p> <table border="1"> <thead> <tr> <th>Event Script</th><th>eventid value</th></tr> </thead> <tbody> <tr> <td>Quality Changed</td><td>qualityChanged</td></tr> <tr> <td>Value Changed</td><td>valueChanged</td></tr> <tr> <td>Alarm Active</td><td>alarmActive</td></tr> <tr> <td>Alarm Cleared</td><td>alarmCleared</td></tr> <tr> <td>Alarm Acknowledged</td><td>alarmAcked</td></tr> </tbody> </table>	Key	Description	eventid	A value representing the type of event script	script	A value representing the content of the script	Event Script	eventid value	Quality Changed	qualityChanged	Value Changed	valueChanged	Alarm Active	alarmActive	Alarm Cleared	alarmCleared	Alarm Acknowledged	alarmAcked	JSON Array of JSON objects. For detailed information, see Tag Event Scripts .
Key	Description																				
eventid	A value representing the type of event script																				
script	A value representing the content of the script																				
Event Script	eventid value																				
Quality Changed	qualityChanged																				
Value Changed	valueChanged																				
Alarm Active	alarmActive																				
Alarm Cleared	alarmCleared																				
Alarm Acknowledged	alarmAcked																				
Alarms																					
Alarms	alarms	<p>Tags have the ability to define any number of alarms. Each alarm is a condition that will be evaluated when the value of the Tag changes. When the condition becomes true, the alarm is said to be active. When it becomes false, the alarm is said to be cleared.</p> <p>For detailed information, see Tag Alarm Properties.</p>	JSON Array of JSON objects. For detailed information, see Tag Alarm Properties .																		
History																					
History Enabled	historyEnabled	Whether the Tag will report its history to the Tags Historian system.	Boolean																		
Storage Provider	historyProvider	Which Tag Historian data store the Tag will target. A particular Tag can only target one history store. For more information, refer to History Providers on the Tag History Gateway Settings page.	String																		
Deadband Style	historicalDeadbandStyle	<p>There are three styles to choose from: Auto, Analog, or Discrete.</p> <p>When set to Auto, this setting will automatically pick from Analog or Discrete, based on the data type of the Tag.</p> <ul style="list-style-type: none"> If the data type of the Tag is set to a float or double, then Auto will use the Analog Style If the data type of the Tag is any other type, then the Discrete style will be used. <p>More information on the Analog and Discrete types can be found on the Configuring Tag History page.</p> <p>Valid values are as follows:</p> <table border="1"> <thead> <tr> <th>Deadband Style</th><th>JSON Name</th></tr> </thead> <tbody> <tr> <td>Auto</td><td>Auto</td></tr> <tr> <td>Analog</td><td>Analog</td></tr> <tr> <td>Discrete</td><td>Discrete</td></tr> </tbody> </table>	Deadband Style	JSON Name	Auto	Auto	Analog	Analog	Discrete	Discrete	String										
Deadband Style	JSON Name																				
Auto	Auto																				
Analog	Analog																				
Discrete	Discrete																				
Deadband Mode	deadbandMode	<p>Defines how the deadband value is used.</p> <ul style="list-style-type: none"> Absolute - The deadband setting is considered to be an absolute value. Percent - The actual deadband is calculated as a percent of the Tag's engineering unit span. <table border="1"> <thead> <tr> <th>Deadband Mode</th><th>JSON Name</th></tr> </thead> <tbody> <tr> <td>Absolute</td><td>Absolute</td></tr> <tr> <td>Percent</td><td>Percent</td></tr> </tbody> </table>	Deadband Mode	JSON Name	Absolute	Absolute	Percent	Percent	String												
Deadband Mode	JSON Name																				
Absolute	Absolute																				
Percent	Percent																				

Historical Deadband	historicalDeadband	A deadband that applies only to historical evaluation.	Numeric																			
Sample Mode	sampleMode	<p>Determines how often a historical record should be collected.</p> <ul style="list-style-type: none"> • On Change - Collects a record whenever the value on the Tag changes. • Periodic - Collects a record based on the Sample Rate and Sample Rate Units properties. • Tag Group - Collects a record based on the Tag Group specified under the Historical Tag Group property. <p>Valid values are as follows:</p> <table border="1"> <thead> <tr> <th>Max Time Between Records Mode</th> <th>JSON Name</th> </tr> </thead> <tbody> <tr> <td>On Change</td> <td>OnChange</td> </tr> <tr> <td>Periodic</td> <td>Periodic</td> </tr> <tr> <td>Tag Group</td> <td>TagGroup</td> </tr> </tbody> </table>	Max Time Between Records Mode	JSON Name	On Change	OnChange	Periodic	Periodic	Tag Group	TagGroup	String											
Max Time Between Records Mode	JSON Name																					
On Change	OnChange																					
Periodic	Periodic																					
Tag Group	TagGroup																					
Sample Rate	historySampleRate	When the Sample Mode property is set to "Periodic", this property (working in conjunction with the Sample Rate Units property) determines how often a record should be collected.	Numeric																			
Sample Rate Units	historySampleRateUnits	When the Sample Mode property is set to "Periodic", this property (working in conjunction with the Sample Rate property) determines the unit of time that will be used in record collection.	<table border="1"> <thead> <tr> <th>Unit of Time</th> <th>JSON Name</th> </tr> </thead> <tbody> <tr> <td>Milliseconds</td> <td>MS</td> </tr> <tr> <td>Seconds</td> <td>SEC</td> </tr> <tr> <td>Minutes</td> <td>MIN</td> </tr> <tr> <td>Hour</td> <td>HOUR</td> </tr> <tr> <td>Day</td> <td>DAY</td> </tr> <tr> <td>Week</td> <td>WEEK</td> </tr> <tr> <td>Month</td> <td>MONTH</td> </tr> <tr> <td>Year</td> <td>YEAR</td> </tr> </tbody> </table>	Unit of Time	JSON Name	Milliseconds	MS	Seconds	SEC	Minutes	MIN	Hour	HOUR	Day	DAY	Week	WEEK	Month	MONTH	Year	YEAR	String
Unit of Time	JSON Name																					
Milliseconds	MS																					
Seconds	SEC																					
Minutes	MIN																					
Hour	HOUR																					
Day	DAY																					
Week	WEEK																					
Month	MONTH																					
Year	YEAR																					
Historical Tag Group	historyTagGroup	When the Sample Mode property is set to "Tag Group", this property determines which Tag Group will be used to collect records.	String																			
Min Time Between Samples	historyTimeDeadband	Minimum time between records. Useful in restricting the number of records collected when the Sample Mode is set to "Tag Change". Prevents multiple consecutive Tag changes from triggering consecutive record collections. Works in conjunctions with the Min Time Units property. The Value is calculated off of the value timestamp.	Integer																			
Min Time Units	historyTimeDeadbandUnits	Units of time to use with the Min Time Between Samples property.	<table border="1"> <thead> <tr> <th>Unit of Time</th> <th>JSON Name</th> </tr> </thead> <tbody> <tr> <td>Milliseconds</td> <td>MS</td> </tr> <tr> <td>Seconds</td> <td>SEC</td> </tr> <tr> <td>Minutes</td> <td>MIN</td> </tr> <tr> <td>Hour</td> <td>HOUR</td> </tr> <tr> <td>Day</td> <td>DAY</td> </tr> <tr> <td>Week</td> <td>WEEK</td> </tr> <tr> <td>Month</td> <td>MONTH</td> </tr> <tr> <td>Year</td> <td>YEAR</td> </tr> </tbody> </table>	Unit of Time	JSON Name	Milliseconds	MS	Seconds	SEC	Minutes	MIN	Hour	HOUR	Day	DAY	Week	WEEK	Month	MONTH	Year	YEAR	String
Unit of Time	JSON Name																					
Milliseconds	MS																					
Seconds	SEC																					
Minutes	MIN																					
Hour	HOUR																					
Day	DAY																					
Week	WEEK																					
Month	MONTH																					
Year	YEAR																					
Max Time between Samples	historyMaxAge	Maximum time between samples. Works in conjunction with the Max Time Units property. If a sample has not been collected by the time range specified by these two properties, then a record will be collected on the next sample interval.	Integer																			
Max Time Units	historyMaxAgeUnits	Maximum time in units is defined as: Milliseconds, Seconds, Minutes, Hours, Days, Weeks, Months, and Years.	<table border="1"> <thead> <tr> <th>Unit of Time</th> <th>JSON Name</th> </tr> </thead> <tbody> <tr> <td>Milliseconds</td> <td>MS</td> </tr> </tbody> </table>	Unit of Time	JSON Name	Milliseconds	MS	String														
Unit of Time	JSON Name																					
Milliseconds	MS																					

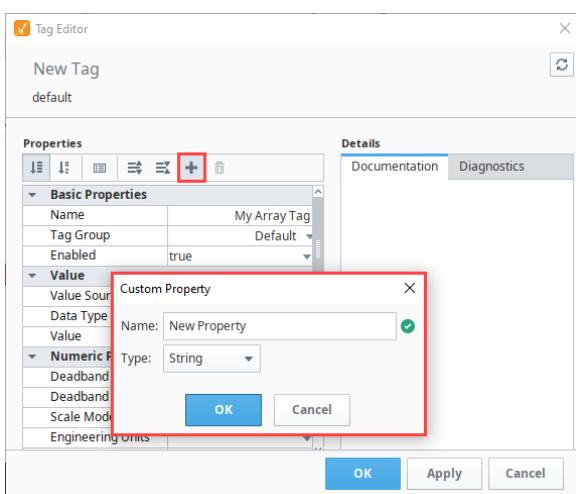
Seconds	SEC
Minutes	MIN
Hour	HOUR
Day	DAY
Week	WEEK
Month	MONTH
Year	YEAR

Custom Tag Properties

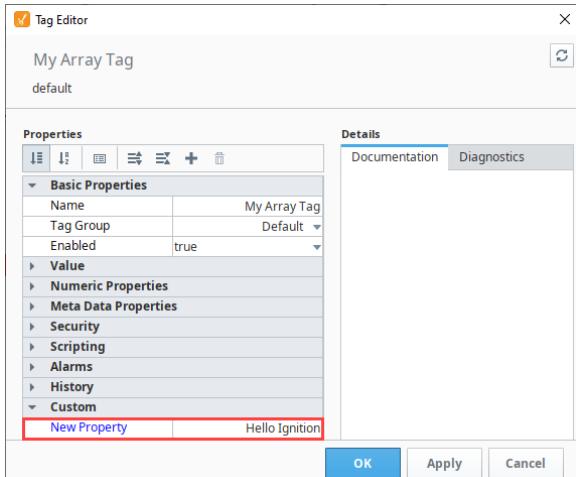
Custom Tag properties allow application designers to configure their own properties on Tags to store unique values on any Tag. Once added, a custom property can be referenced like any other Tag property via bindings, expressions, and scripts.

Both the Perspective and Vision visualization systems can bind to custom properties. In the following example, we already have Array Tag. Now let's add a custom property.

1. Open an existing Tag and click on the Add  icon in the Tag Editor.
2. This opens the Custom Property dialog box. Enter a **Name** for your Tag, select the **Data Type**, and click **OK**.



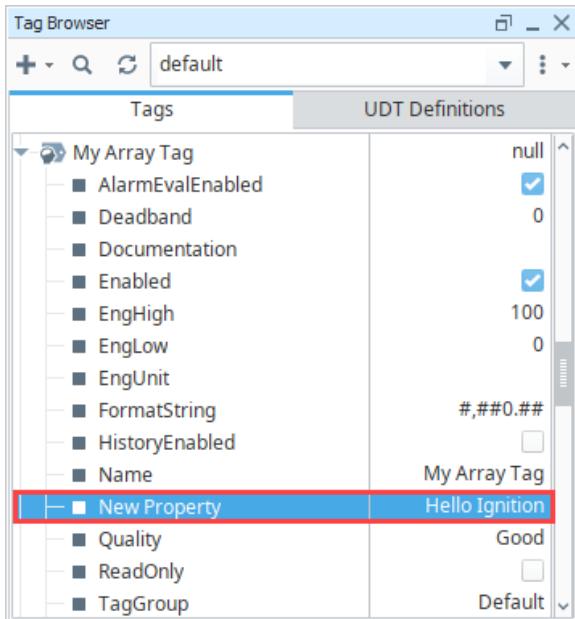
3. Scroll down to the bottom of your Tag properties and you'll see your Custom Property. We added the value "Hello Ignition".



Custom Tag Properties

[Watch the Video](#)

4. When you open your Tag Browser and expand the My Array Tag, you'll see your Custom Property.



Vision Client Tags

Client Tags have the ability to be used as either Expression or SQL Query Tags. There is an [Expression/SQL](#) page in the Tag editor that allows you to select what type it is.

Query/Expression Attributes												
Property	Binding /Scripting Name	Description	Data Type	Applicable Tag Type								
OPC Server	OPCWriteBackServer	The server against which to subscribe the data point.	String	Query, Expression								
OPC Item Path	OPCWriteBackItemPath	The path to subscribe to on the server.	String	Query, Expression								
Query	Expression	Text area to build your query or expression. This is the code used by the Tag: either a SQL Query for Query Tags, or an Expression for Expression Tags.	String This is the code used by the Tag: either a SQL Query for Query Tags, or an Expression for Expression Tags.	Query, Expression								
Query Type	QueryType	When the TagType property is set to 1, this property determines if the Tag should be a Memory, Expression, or Query Tag.	Integer <table border="1"> <thead> <tr> <th>Tag Type</th> <th>Integer Value</th> </tr> </thead> <tbody> <tr> <td>Memory Tag</td> <td>0</td> </tr> <tr> <td>Expression Tag</td> <td>1</td> </tr> <tr> <td>Query Tag</td> <td>2</td> </tr> </tbody> </table>	Tag Type	Integer Value	Memory Tag	0	Expression Tag	1	Query Tag	2	Query, Expression, Memory
Tag Type	Integer Value											
Memory Tag	0											
Expression Tag	1											
Query Tag	2											
Datasource	SQLBindingDataSource	The default data source of the Tag provider.	String	Query								

Related Topics ...

- [Types of Tags](#)
- [Alarm Event Properties Reference](#)
- [Tag Event Scripts](#)

- [Configuring Tag Historian](#)

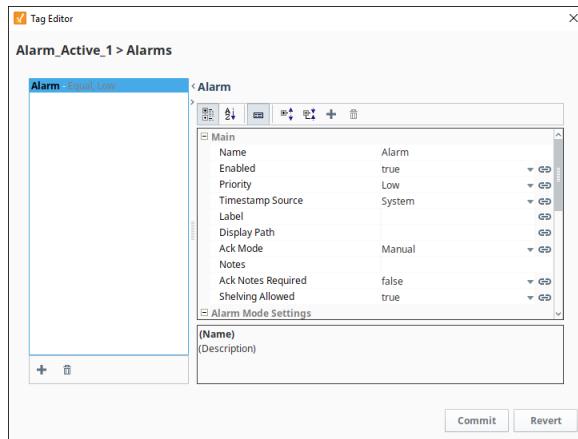
In This Section ...

Tag Alarm Properties

Tags have the ability to define any number of alarms. Each alarm is a condition that will be evaluated when the value of the Tag changes. When the condition becomes true, the alarm is said to be active. When it becomes false, the alarm is said to be cleared.

To add an alarm to a Tag, click the **Edit** icon next to Alarms. The Alarms screen is displayed.

Click the **Add** icon to add an alarm. The table below lists all the Tag Alarm property settings.



On this page ...

- [Alarm Property Reference](#)
 - [Alarms in Scripting](#)
 - [Reference Table](#)
- [Runtime Tag Alarm Properties](#)
- [Binding](#)



Tag Scripts

[Watch the Video](#)

Alarm Property Reference

The table in this section provides a description of alarm properties.

Alarms in Scripting

When interacting with the tags system in via scripting, such as with the `system.tag.configure` function, alarms are represented as a JSON array of JSON objects, where each object contains the configurations for a single alarm. Thus, any scripting names here are assumed to exist under the alarms array.

Reference Table

Property Name	JSON /Scripting Name	Description	Datatype	Applicable Tag Type												
Main																
Name	name	Identifier of the alarm. Combined with the location of the alarm, this will be the unique alarm ID. For dynamic values, used Label or Display Path.	String	OPC, Query, Expression, Derived, Reference, Memory												
Enabled	enabled	This boolean determines whether or not the alarm will be evaluated. A disabled alarm's condition will not be evaluated, and thus will not generate any alarm events.	Boolean	OPC, Query, Expression, Derived, Reference, Memory												
Priority	priority	The priority or severity of this alarm. Alarm priorities can be examined by many other systems in Ignition, including the visualization modules, pipelines, and even scripting.	String <table border="1"><tr><td>Priority</td><td>JSON Name</td></tr><tr><td>Diagnostic</td><td>Diagnostic</td></tr><tr><td>Low</td><td>Low</td></tr><tr><td>Medium</td><td>Medium</td></tr><tr><td>High</td><td>High</td></tr><tr><td>Critical</td><td>Critical</td></tr></table>	Priority	JSON Name	Diagnostic	Diagnostic	Low	Low	Medium	Medium	High	High	Critical	Critical	OPC, Query, Expression, Derived, Reference, Memory
Priority	JSON Name															
Diagnostic	Diagnostic															
Low	Low															
Medium	Medium															
High	High															
Critical	Critical															

Timestamp Source	timestampSource	Indicates where the timestamp for the alarm event should come from: the system time of when the event was generated (i.e., the Gateway's time), or the timestamp of the value that triggered the event (i.e., the timestamp of the value from the OPC server).	String	<table border="1"> <thead> <tr> <th>Timestamp Source</th><th>JSON Name</th><th>Int Value</th></tr> </thead> <tbody> <tr> <td>System</td><td>System</td><td>0</td></tr> <tr> <td>Value</td><td>Value</td><td>1</td></tr> </tbody> </table>	Timestamp Source	JSON Name	Int Value	System	System	0	Value	Value	1	OPC, Query, Expression, Derived, Reference, Memory			
Timestamp Source	JSON Name	Int Value															
System	System	0															
Value	Value	1															
Label	label	An optional name that will be used for display purposes. Provides a dynamic alternative to the static name property. If left blank, the name will be used.	String		OPC, Query, Expression, Derived, Reference, Memory												
Display Path	displayPath	Optional text that will be used for display and browsing purposes. If this is blank, this property will show the path to the Tag and the name of the alarm instead.	String		OPC, Query, Expression, Derived, Reference, Memory												
Ack Mode	ackMode	Dictates how acknowledgement works for the alarm.	String	<table border="1"> <thead> <tr> <th>Ack Mode</th><th>JSON Name</th><th>Int Value</th></tr> </thead> <tbody> <tr> <td>Unused</td><td>Unused</td><td>0</td></tr> <tr> <td>Auto</td><td>Auto</td><td>1</td></tr> <tr> <td>Manual</td><td>Manual</td><td>2</td></tr> </tbody> </table>	Ack Mode	JSON Name	Int Value	Unused	Unused	0	Auto	Auto	1	Manual	Manual	2	OPC, Query, Expression, Derived, Reference, Memory
Ack Mode	JSON Name	Int Value															
Unused	Unused	0															
Auto	Auto	1															
Manual	Manual	2															
Notes	notes	A place for any free-form documentation about the alarm that can be displayed to users.	String		OPC, Query, Expression, Derived, Reference, Memory												
Ack Notes Required	ackNotesReqd	If this setting is true, the operators will be required to provide some explanation when the alarm is acknowledged.	Boolean		OPC, Query, Expression, Derived, Reference, Memory												
Shelving Allowed	shelvingAllowed	If this setting is false, the shelving feature will be unavailable for this alarm.	Boolean		OPC, Query, Expression, Derived, Reference, Memory												

Alarm Mode Settings

Mode	mode	This setting controls what condition this alarm is evaluating. The available modes are as follows: <ul style="list-style-type: none">• Equal - Active when the Tag's value equals the alarm's setpoint.• Not Equal - Active when the Tag's value does not equal the alarm's setpoint.• Above Setpoint - Active when the Tag's value is above the alarm's setpoint.• Below Setpoint - Active when the Tag's value is below the alarm's setpoint.• Between Setpoints - Active when the Tag's value is between the low and high setpoints. If any change is true, an event will be generated for each value change between the setpoints.• Outside Setpoints - Active when the Tag's value falls outside the low and high setpoints. If any change is true, an event will be generated for each value change outside the setpoints.• Out of Range - The same as Outside Setpoints, but uses the Tag's Engineering High and Engineering Low as the high and low setpoints.• Bad Quality - Active if the Tag value becomes a bad quality, for example, on communication loss.• Any Change - An alarm event is generated every time the Tag value changes. <p>Note: This alarm will never be "active" because each active event is paired with a matching clear event, instantly.</p> <ul style="list-style-type: none">• Bit State - This alarm mode is used to alarm when a specific bit out of an integer Tag becomes high. You must specify which bit position to use, with zero being the least significant bit. The On Zero property is used to invert the logic and alarm when the bit is low.• On Condition - This free-form alarm mode is used for when you want to specify the condition using an expression or another Tag. To do this, bind the "Is Active" property to an appropriate expression or Tag.	String	<table border="1"> <thead> <tr> <th>Mode</th><th>JSON Name</th></tr> </thead> <tbody> <tr> <td>Equal</td><td>Equality</td></tr> <tr> <td>Not Equal</td><td>Inequality</td></tr> <tr> <td>Above Setpoint</td><td>AboveValue</td></tr> <tr> <td>Below Setpoint</td><td>BelowValue</td></tr> <tr> <td>Between Setpoints</td><td>BetweenValues</td></tr> <tr> <td>Outside Setpoints</td><td>OutsideValues</td></tr> <tr> <td>Out of Range</td><td>OutOfEngRange</td></tr> <tr> <td>Bad Quality</td><td>BadQuality</td></tr> <tr> <td>Any Change</td><td>AnyChange</td></tr> <tr> <td>Bit State</td><td>Bit</td></tr> <tr> <td>On Condition</td><td>OnCondition</td></tr> </tbody> </table>	Mode	JSON Name	Equal	Equality	Not Equal	Inequality	Above Setpoint	AboveValue	Below Setpoint	BelowValue	Between Setpoints	BetweenValues	Outside Setpoints	OutsideValues	Out of Range	OutOfEngRange	Bad Quality	BadQuality	Any Change	AnyChange	Bit State	Bit	On Condition	OnCondition	OPC, Query, Expression, Derived, Reference, Memory
Mode	JSON Name																												
Equal	Equality																												
Not Equal	Inequality																												
Above Setpoint	AboveValue																												
Below Setpoint	BelowValue																												
Between Setpoints	BetweenValues																												
Outside Setpoints	OutsideValues																												
Out of Range	OutOfEngRange																												
Bad Quality	BadQuality																												
Any Change	AnyChange																												
Bit State	Bit																												
On Condition	OnCondition																												
Setpoint /Low Setpoint	setpointA	Used to determine if the alarm is active by comparing this value to the tag value. Some modes under the Mode property allow for multiple setpoints (i.e., a low setpoint and a high setpoint). In these cases, this property is considered to be the Low setpoint.	Numeric		OPC, Query, Expression, Derived, Reference, Memory																								
Inclusive	inclusiveA	If true, the Setpoint value is used inclusively for the condition to alarm.	Boolean		OPC, Query, Expression, Derived, Reference, Memory																								
High Setpoint	setpointB	The high value used to initiate an alarm when the alarm mode calls for two setpoints. Available for modes: Between Setpoint, Outside Setpoints.	Numeric		OPC, Query, Expression, Derived, Reference, Memory																								
High Inclusive	inclusiveB	If true, the High Setpoint value is used inclusively for the condition to alarm.	Boolean		OPC, Query,																								

		Available for modes: Between Setpoint, Outside Setpoints.		Expression, Derived, Reference, Memory
Any Change	anyChange	If true, will alarm on each value change given the alarm mode conditions are met. Note: This alarm will never be "active" because each active event is paired with a matching clear event, instantly. Available for modes: Above Setpoint, Below Setpoint, Between Setpoints, and Outside Setpoints.	Boolean	OPC, Query, Expression, Derived, Reference, Memory
On Zero	bitOnZero	If true, will alarm when the specified bit is not high (when the bit is 0).	Boolean	OPC, Query, Expression, Derived, Reference, Memory
Bit Position	bitPosition	The position of the bit, starting at 0 that will be watched. Available for modes: Bit State.	Numeric	OPC, Query, Expression, Derived, Reference, Memory
Is Active	activeCondition	When this property is active, the alarm will be active. Typically has a binding of some sort that will be used to determine when the alarm goes active. If the expression evaluates to True, the alarm is active. If the expression evaluates to False, the alarm is not active.	Boolean	OPC, Query, Expression, Derived, Reference, Memory

Deadbands and Time Delays

Deadband	deadband	The value for the deadband, interpreted according to the Deadband mode. Note: All alarms are only evaluated after the Tag's value changes, which means that the Tag's own deadband will be considered first. When the deadband is positive, an active alarm condition needs to clear its setpoint(s) by the amount of the deadband for the alarm to clear. For example, suppose you had a Between Setpoints alarm with a low setpoint of 50 and a high setpoint of 70, and with a deadband of 2. The alarm will go active if the value is between 50 and 70, but will only clear if the value falls below 48 or rises above 72.	Numeric	OPC, Query, Expression, Derived, Reference, Memory						
Deadband Mode	deadbandMode	Defines how the deadband value is used. <ul style="list-style-type: none">• Absolute - The deadband setting is considered to be an absolute value.• Percent - The actual deadband is calculated as a percent of the Tag's engineering unit span.	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>Alarming Deadband Mode</th> <th>JSON Name</th> </tr> <tr> <td>Absolute</td> <td>Absolute</td> </tr> <tr> <td>Percent</td> <td>Percent</td> </tr> </table>	Alarming Deadband Mode	JSON Name	Absolute	Absolute	Percent	Percent	OPC, Query, Expression, Derived, Reference, Memory
Alarming Deadband Mode	JSON Name									
Absolute	Absolute									
Percent	Percent									
Active Delay	timeOnDelaySeconds	The time, in seconds, before the alarm will be considered active after the alarm's condition becomes true. Also known as a "rising edge time deadband."	Numeric	OPC, Query, Expression, Derived, Reference, Memory						
Clear Delay	timeOffDelaySeconds	The time, in seconds, before an active alarm will be considered clear after the alarm's condition becomes false. Also known as a "falling edge time deadband."	Numeric	OPC, Query, Expression, Derived, Reference, Memory						

Notification Properties

Active Pipeline	activePipeline	The name of an alarm notification pipeline to put this alarm into when it becomes active in order to send out active alarm messages. Many alarms may share a single pipeline.	String	OPC, Query, Expression, Derived, Reference, Memory
Clear Pipeline	clearPipeline	The name of an alarm notification pipeline to put this alarm into when it becomes cleared in order to send out cleared messages.	String	OPC, Query, Expression, Derived, Reference, Memory
Ack Pipeline	ackPipeline	The name of the alarm notification pipeline to put this alarm into when the alarm has been acknowledged.	String	OPC, Query, Expression, Derived, Reference, Memory

Email Notification Properties

Custom Subject	CustomEmailSubject	A string that will be used as the subject line of an email notification message. If blank, the message settings defined on the notification block that sent the email out will be used instead.	String	OPC, Query, Expression, Derived, Reference, Memory
Custom Message	CustomEmailMessage	A string that will be used as the body of this alarm's email notification message. If blank, the message settings defined on the notification block that sent the email out will be used instead.	String	OPC, Query, Expression, Derived, Reference, Memory

SMS Notification Properties

Custom Message	CustomSmsMessage	If specified, will be used for the SMS message. If blank, the message defined in the notification block will be used.	String	OPC, Query, Expression, Derived, Reference, Memory
----------------	------------------	---	--------	--

Associated Data

User Defined Data		Associated Data are custom alarm properties that can be added to any alarm. These properties will often be bound to other Tags that represent associated contextual data that may be related to the alarm. A snapshot of the values of these properties will be taken when the alarm becomes active. These values will be attached to the alarm event as it moves through the rest of the alarming system, meaning that the values will be available from the alarm status system, the alarm journal system, and in the alarm notification system.	String	OPC, Query, Expression, Derived, Reference, Memory
-------------------	--	--	--------	--

Runtime Tag Alarm Properties

There are a number of very useful Runtime Alarm Tag Properties that expose helpful information on the count of alarms in various states, priority, and shelved alarms.

Property Name	Description
ActiveAckCount	The number of alarms on the Tag that are both Active and Acknowledged.
ActiveUnackCount	The number of alarms on the Tag that are both Active and Unacknowledged.
ClearUnackCount	The number of alarms on the Tag that are both Clear and Unacknowledged.
HasActive	True, if the Tag has at least one Active alarm. False, if there are zero alarms.
HasUnacknowledged	True, if the Tag has at least one Unacknowledged alarm. False, if there are zero Unacknowledged alarms.
HighestAckedName	The Name of the highest Acknowledged alarm, ranked by Priority.
HighestAckedPriority	The highest Priority of all Acknowledged alarms on the Tag.
HighestActiveName	The Name of the highest Active alarm, ranked by Priority.
HighestActivePriority	The highest Priority of all Active alarms on the Tag.
HighestUnackedName	The Name of the highest Unacknowledged alarm, ranked by Priority.
HighestUnackedPriority	The highest Priority of all Unacknowledged alarms on the Tag.
LastActiveTime	A timestamp representing the last time an alarm went Active on the Tag.
ShelvedCount	The number of currently shelved alarms on the Tag.

Binding

Many alarm properties are bindable, which means they can be bound to other Tags in the system, or expressions. For example, you might bind the enabled property to another Tag which represents whether or not your process is running, thereby, disabling the alarm when production is stopped. Another example is you might bind the setpoint of an alarm to a Tag that operators can manipulate, thereby, letting the setpoint be changed at runtime. For more information, see [Configuring Alarms](#).

To bind an alarm property of a Tag, click on the binding  icon, and the binding UI will slide in from the right.

Tag Editor

fanSpeed > Alarms

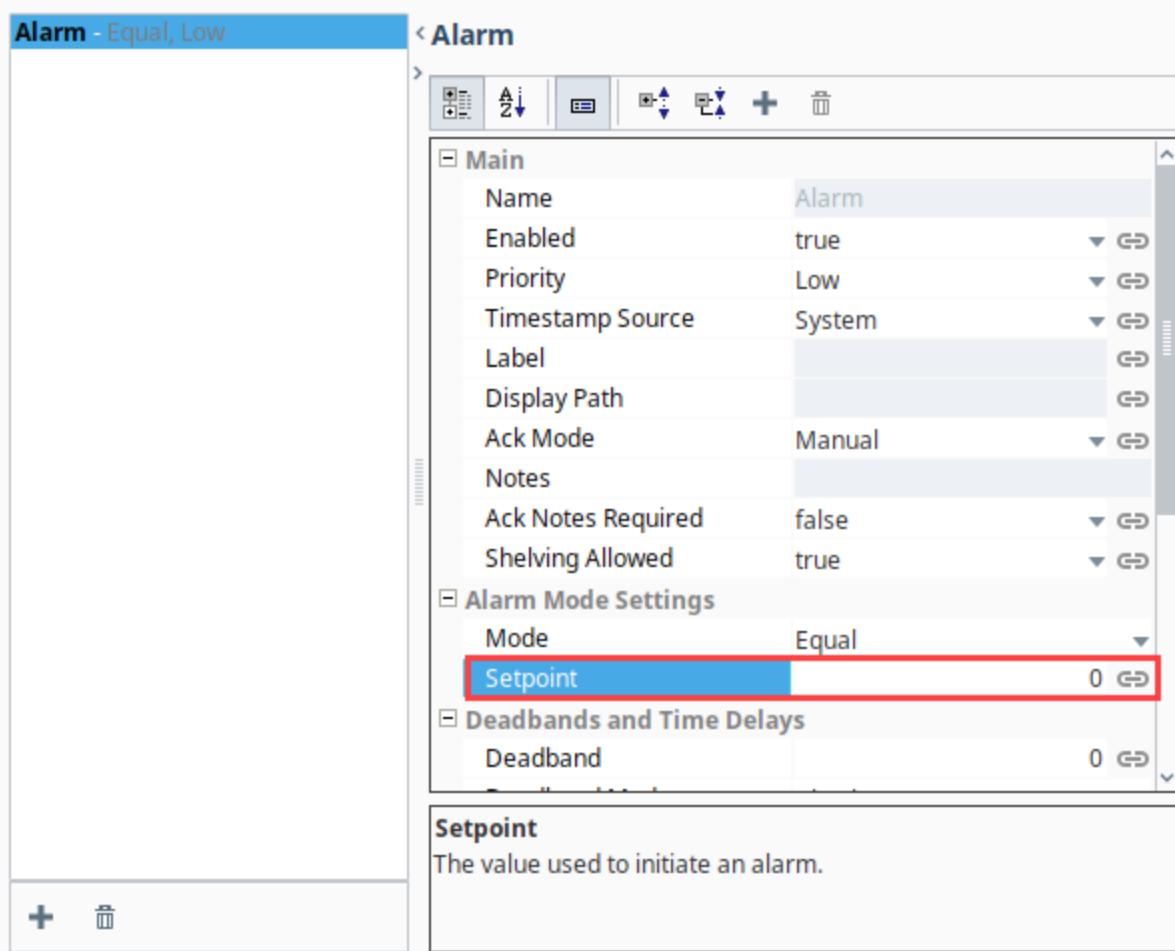
Alarm - Equal, Low

Alarm

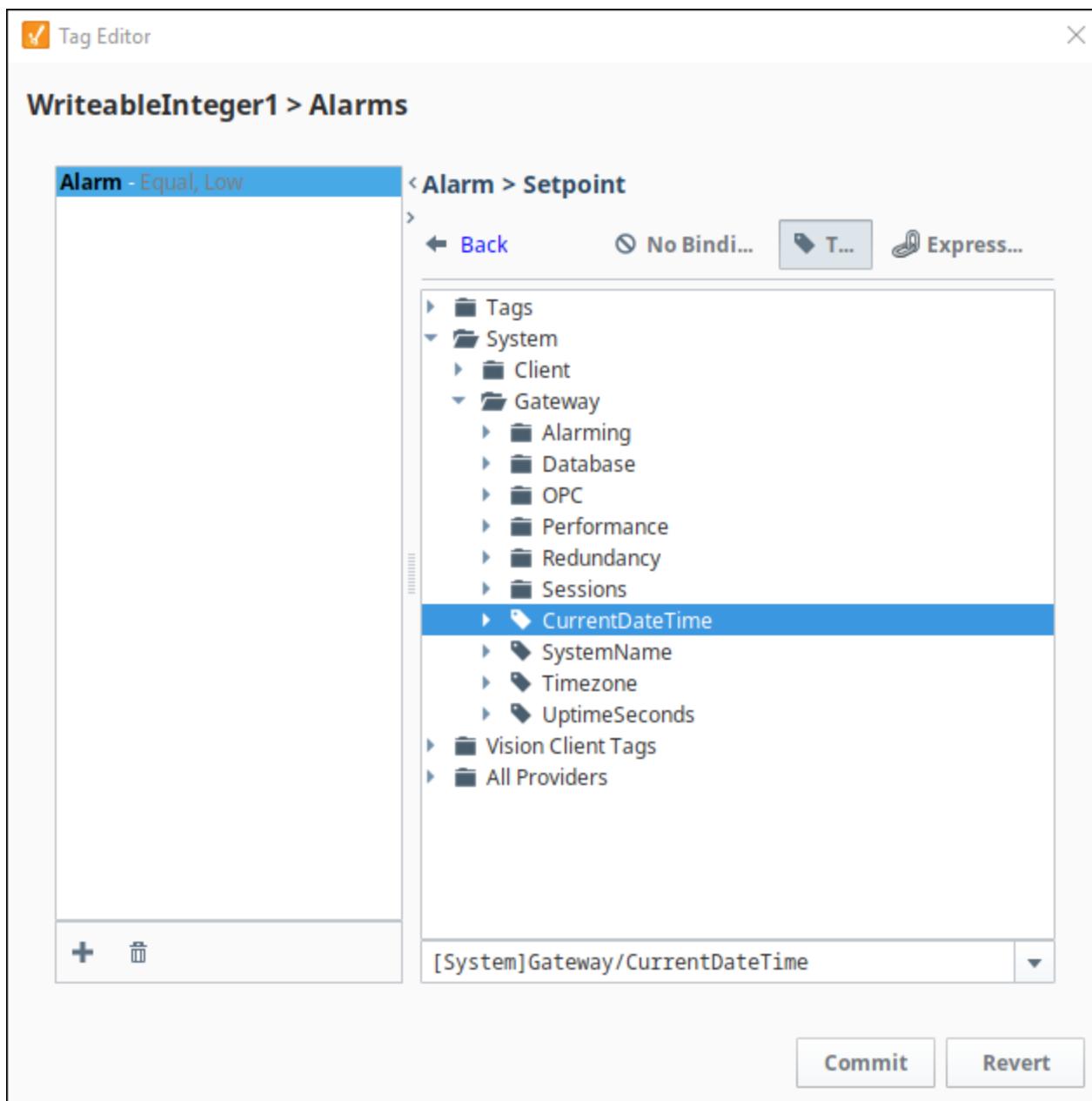
Main	
Name	Alarm
Enabled	true
Priority	Low
Timestamp Source	System
Label	
Display Path	
Ack Mode	Manual
Notes	
Ack Notes Required	false
Shelving Allowed	true
Alarm Mode Settings	
Mode	Equal
Setpoint	0
Deadbands and Time Delays	
Deadband	0

Setpoint
The value used to initiate an alarm.

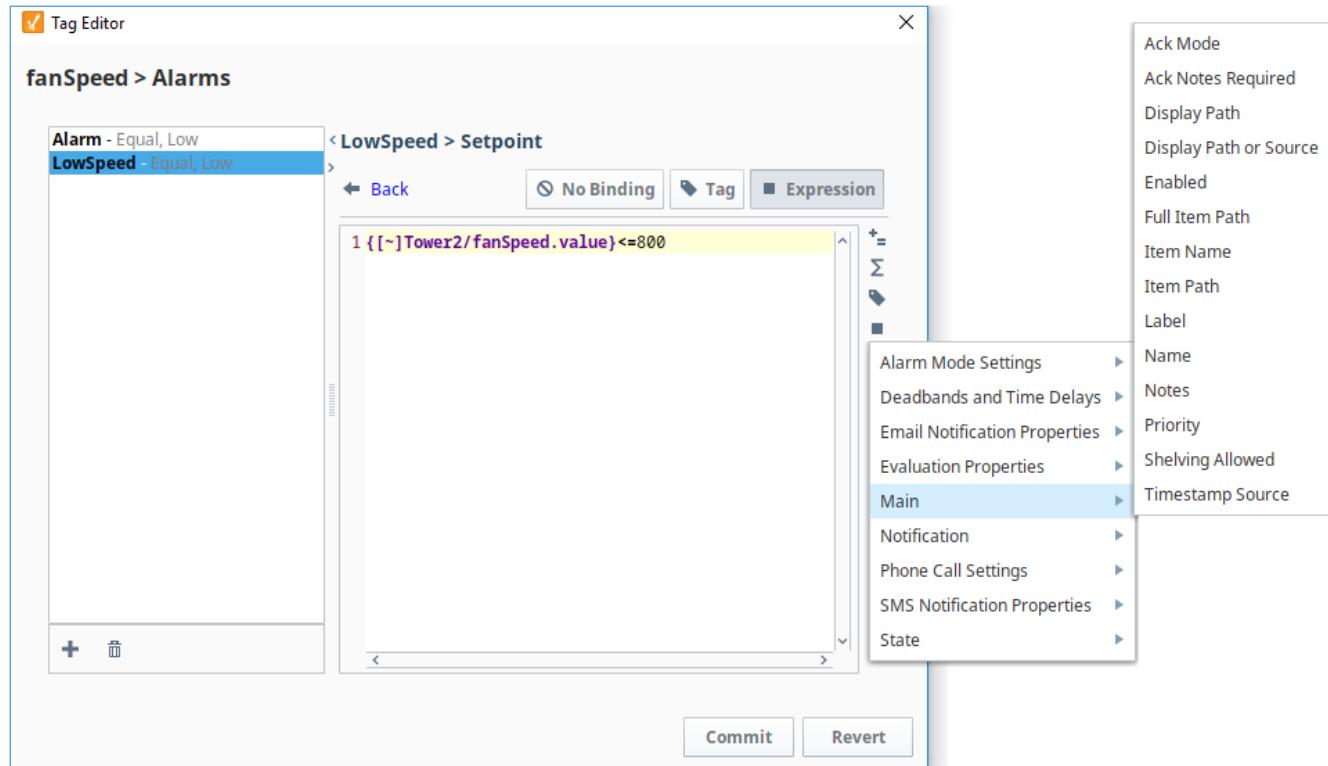
Commit **Revert**



From here, you can select the binding type (No Binding, Tag, Expression, or UDT Parameter, if applicable).



Binding to an Expression can reference many useful values such as the Tag's value and other settings of the alarm.

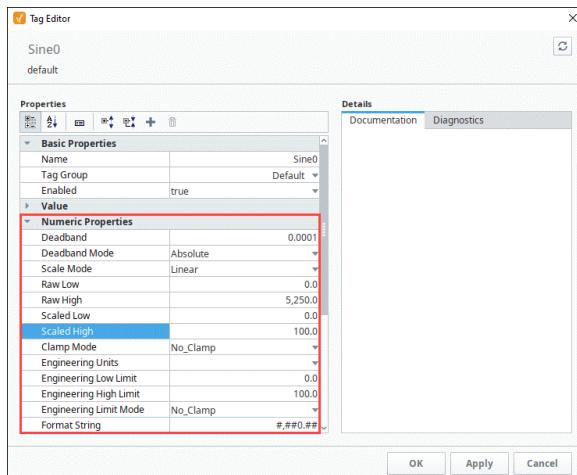


When you configured the binding to your liking, click the **Commit** button.

Tag Scaling Properties

Configuring a Tag's scaling will condition the data for use within the Ignition Designer. Scaling will take the raw PLC value driving a Tag, do some math, and use the resulting value as the value of that Tag. Scaling works both ways. When you write to that Tag, Ignition will scale it in the opposite direction before writing to the PLC.

For example, if the capacity of a tank is 5250 gallons, but the tank's fill level is better represented in the Designer as percentage of 0 through 100, configuring the Tag's scaling property, will result in the Tag displaying 0 through 100, while the actual tank fill moving is between 0 and 5250 gallons. For this example, you can double-click on the Tag to open the **Tag Editor**, and expand **Numeric Properties** to configure the scaling of the Tag. When scaling between a Raw Low and High, and Scaled Low and High, select the **Linear** Scale Mode. So what Ignition is actually doing, is setting up the calculations behind the scenes to scale the value appropriately.



On this page ...

- [Linear Scaling](#)
- [Square Root Scaling](#)
- [Exponential Filter Scaling](#)
- [Bit Inversion Scaling](#)
- [Scaling Examples](#)
 - [Simple Divide by Ten](#)
 - [4-20 Milliamp Signal to Percent](#)
 - [4-20 Milliamp Signal to Gallons](#)



Tag Scaling

[Watch the Video](#)



If you are using scaling...

The numbers and units don't have to match-up. Scaling is straight from one number to another number. It doesn't matter what the units are, and it doesn't matter what the conversion is. What is important, is that the data type of the Tag must match the data type of the scale value (i.e., dividing an integer in the PLC by 10 probably means your Ignition Tag should be a float).

Scaling is not available on Memory Tags: Memory Tags are not driven by an external source such as a PLC or SQL query, so scaling will never be applied. In these scenarios, it is recommended to scale the mechanism that is writing to the Memory Tag instead. Numerical properties of Tags can be scaled allowing automatic bi-directional conversion outside of the PLC. Scaling types include **Linear** scaling, **Square Root** scaling, and **Exponential Filter** scaling. The numerical properties are available to **OPC**, **Expression**, **Database**, and **Client** Tags whose data types are numeric. For a complete list of all of the Tag Scaling properties, see [Tag Properties](#).

Linear Scaling

The value will be scaled linearly between the low and high values, and clamped as appropriate. The linear equation is:

```
ScaledValue = S * (Value-RL) / R + SL
```

Where:

$$\begin{aligned} S &= (\text{ScaledHigh}-\text{ScaledLow}) \\ R &= (\text{RawHigh} - \text{RawLow}) \\ RL &= \text{RawLow} \\ SL &= \text{ScaledLow} \end{aligned}$$

Square Root Scaling

The equation for square root scaling is:

```
ScaledValue = S * ((Value-RL)/R) + SL
```

Where:

S = (ScaledHigh-ScaledLow)
R = (RawHigh - RawLow)
RL = RawLow
SL = ScaledLow

Exponential Filter Scaling

This mode implements a simple linear recursive filter to smooth values. The scale factor corresponds to the weight of the smoothing effect, and is a value between 0.0 and 1.0. The smaller the factor, the greater the degree of smoothing.

The equation for the filter is:

```
y(t) = (1-f)*y(t-1)+f*x(t)
```

Where:

y(t) = the output at time t
y(t-1) = the previous output
x(t) = the input value (current value)
f = the scale factor, with $0.0 \leq f \leq 1.0$

Note: Only **good** quality values are considered for the filter. **Bad** quality values are ignored.

Bit Inversion Scaling

This simple scaling mode will generate the complement of a binary value. If the current value is coming in as 0001_0101 (21), this will return a 1110_1010 (-22) instead. A popular use for this scale mode is that it can be used to invert modbus values if your device stores them in reverse bit order. Note that Bit Inversion Scaling uses a little-endian format.

Scaling Examples

Simple Divide by Ten

This is common when storing a single decimal point of precision as an Integer in the PLC. This is to save space by not using a Float type.

Raw Low: 0.0
Raw High: 100.0
Scaled Low: 0.0
Scaled High: 10.0

4-20 Milliamp Signal to Percent

This is common when using a simple pressure sensor. The sensor is calibrated to send 4 millamps (minimum value) when the tank is empty, and 20 millamps (maximum value) when the tank is full.

Raw Low: 4.0
Raw High: 20.0
Scaled Low: 0.0
Scaled High: 100.0

4-20 Milliamp Signal to Gallons

This is common when using a simple pressure sensor. The sensor is calibrated to send 4 millamps (minimum value) when the tank is empty, and 20 millamps (maximum value) when the tank is full (5000 gallons).

Note: There is no direct conversion between amps and gallons. In scaling it doesn't matter.

Raw Low: 4.0

Raw High: 20.0
Scaled Low: 0.0
Scaled High: 5000.0

Related Topics ...

- [Exporting and Importing Tags](#)

Tag Security Properties

Tag security is often the best way to configure security for data access. By defining security on a Tag, you affect the Tag across wherever it is used, as opposed to configuring component security on each component that displays or controls that Tag.

There are three properties on Tags that can restrict access.

- Read Permissions: Defines the security levels required in order to read values from a Tag
- Read Only: Defines whether a Tag is read-only or writable
- Write Permissions: Defines the security levels required in order to write values to a Tag

Users with specific roles and zones can be given read/write access to a Tag, while other users with other roles are excluded from modifying the Tag.

If a user opens a Perspective view or a Vision client window that has components that are bound to a Tag they do not have permissions for, the user will see an overlay on top of the component. For more information, see [Tag Quality and Overlays](#). The following example shows a tank displayed in a session, but the user does not have read permission for the Tag it is bound to.



On this page ...

- [Read Only Security](#)
- [Read and Write Permissions](#)
 - [Read Permissions](#)
 - [Write Permissions](#)
- [Using Security Levels](#)

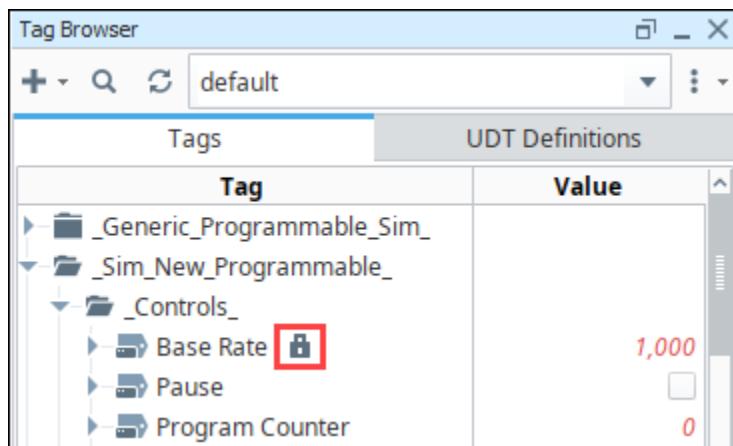


Tag Security

[Watch the Video](#)

Read Only Security

When a Tag is set to read only, a Lock  icon is displayed next to the Tag in the Tag Browser.



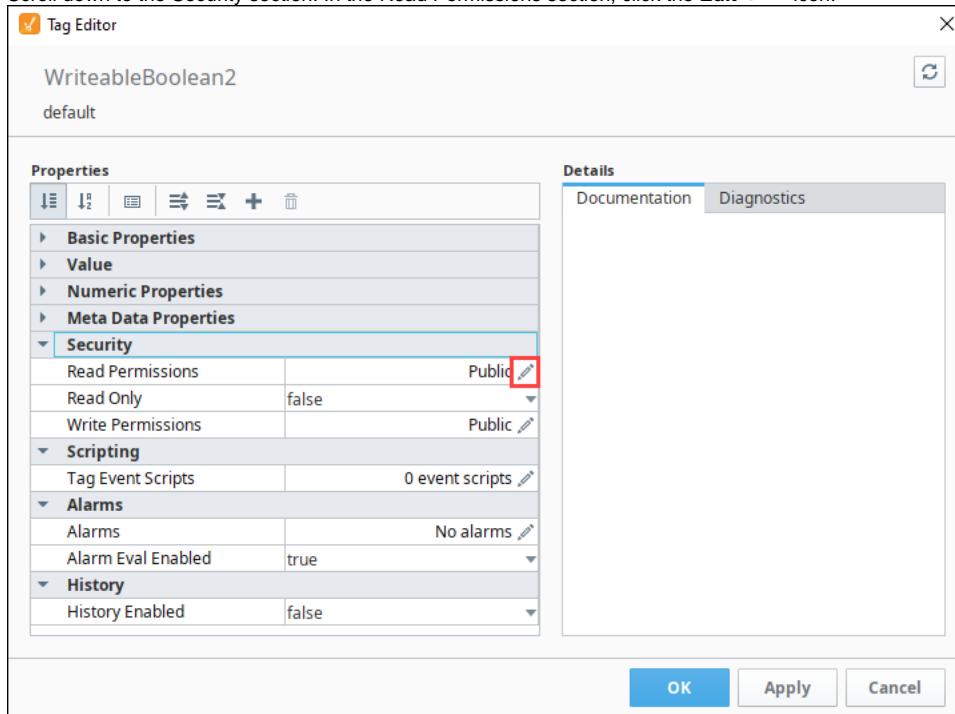
Read and Write Permissions

Instead of making a tag Read Only for all users, you can conditionally provide read and write access based on [Security Levels](#). Doing so involves adjusting the security settings on the tag in question. The checkbox tree you are presented with will show you all of the security levels configured in the Gateway Config > Security > [Security Levels](#) page.

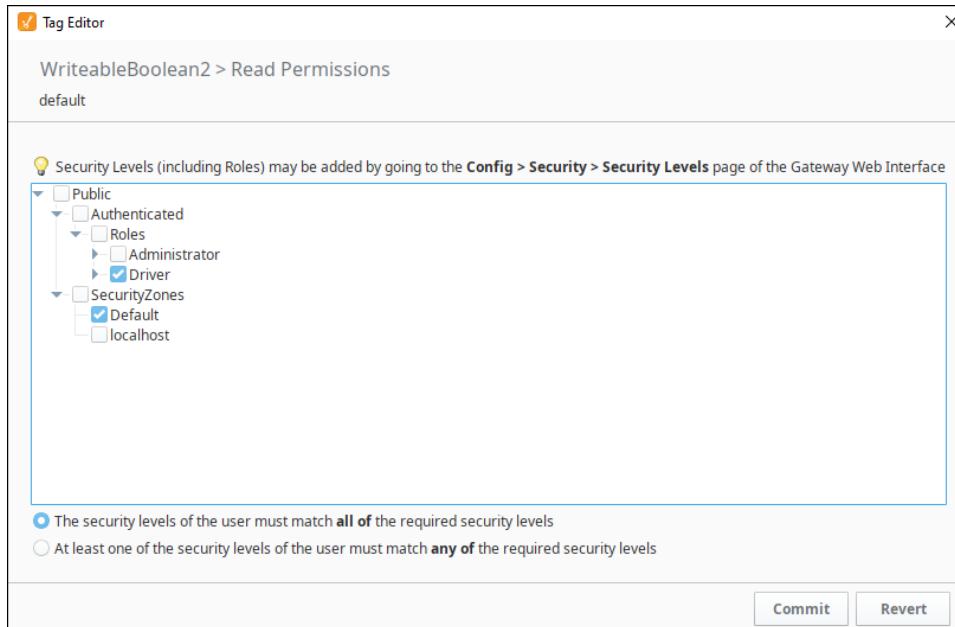
Read Permissions

Read permissions define the security levels required in order to read values from a Tag. By default, Tags have Read Permissions set to "Public". You can change the Read security using the Tag Browser in the Designer.

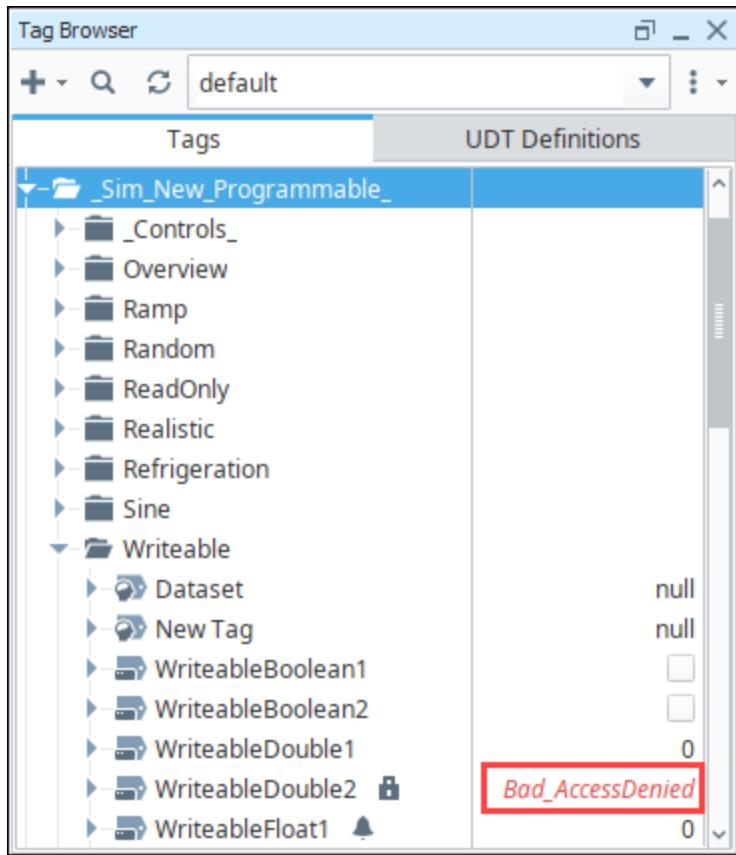
1. In the Tag Browser, right-click on the **Tag**, and select the **Edit** icon.
2. Scroll down to the Security section. In the Read Permissions section, click the **Edit** icon.



3. On the screen, choose the security levels you want to have Read permissions for this Tag. In this example, only users with role of "Driver" will be able to see the Tag value.



4. Click **Commit** to accept the settings.
5. Click **OK** to save the changes to the Tag.
6. If you are logged in as a user other than Driver, you will now see the "Bad_AccessDenied" in the Tag Browser instead of the Tag's value.

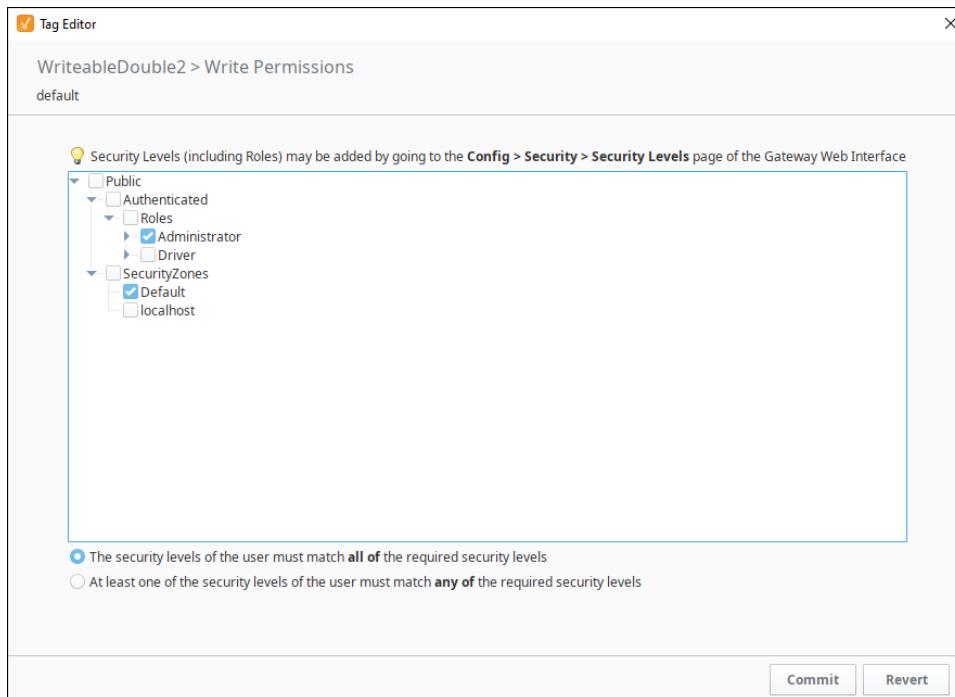


Write Permissions

Write permissions define the security levels required in order to read values from a Tag. By default, Tags have Write Permissions set to "Public". You can change the Write security using the Tag Browser in the Designer.

1. In the Tag Browser, right-click on the **Tag**, and select the **Edit** icon.
2. Scroll down to the Security section. In the Write Permissions section, click the **Edit** icon.

3. On the screen, choose the security levels you want to have write permissions for this Tag. In this example, only users with role of Administrator will be able to write to the Tag value.



Using Security Levels

In addition to setting up security on individual Tags, you can set up security policies specific to each Security Zone. This is useful in cases where you wanted to make all tags in a provider read only from network locations. Tag Access is one of the options for a [Security Zones](#) page for more details.

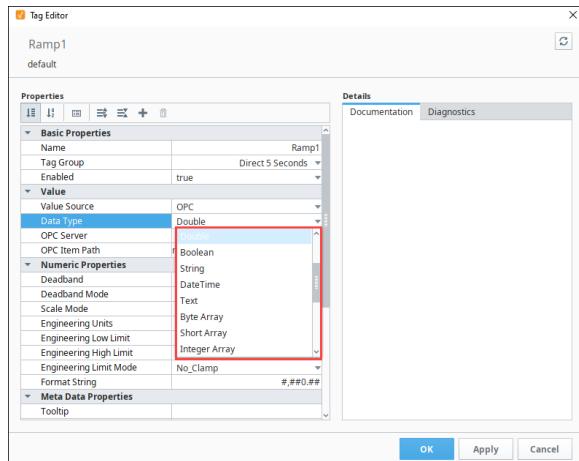
Related Topics ...

- [Audit Log and Profiles](#)
- [Quality Overlays](#)
- [Permission Properties](#)

Tag Data Types

This page details the different data types that can be applied to Standard Tags.

The data type of a Tag is determined by the Data Type property, which is accessible from the Tag Editor. The Tags system will attempt to coerce any raw incoming value (for example, from OPC or a SQL query) into the desired type.



On this page ...

- [Array and Dataset Data Types](#)
 - [Array Tags](#)
 - [Dataset Tags](#)
 - [Dataset Tag Example](#)



Array and Dataset Tags

[Watch the Video](#)

The following table lists all the data types available for Tags in Ignition.

Data Type	String Value	Integer Value
Byte	Int1	0
Short	Int2	1
Integer	Int4	2
Long	Int8	3
Float	Float4	4
Double	Float8	5
Boolean	Boolean	6
String	String	7
DateTime	DateTime	8
Text	Text	10
Byte Array	Int1Array	17
Short Array	Int2Array	18
Integer Array	Int4Array	11

Long Array	Int8Array	12
Float Array	Float4Array	19
Double Array	Float8Array	13
Boolean Array	BooleanArray	14
String Array	StringArray	15
DateTime Array	DateTimeArray	16
Binary Data	ByteArray	20
Dataset	DataSet	9
Document	Document	29

Array and Dataset Data Types

The Array and Dataset data types available on Tags allow for multiple data points to be stored in a single Tag. Configuring a Tag as an array or dataset is as easy as changing the data type in the Tag Editor.

Array Tags

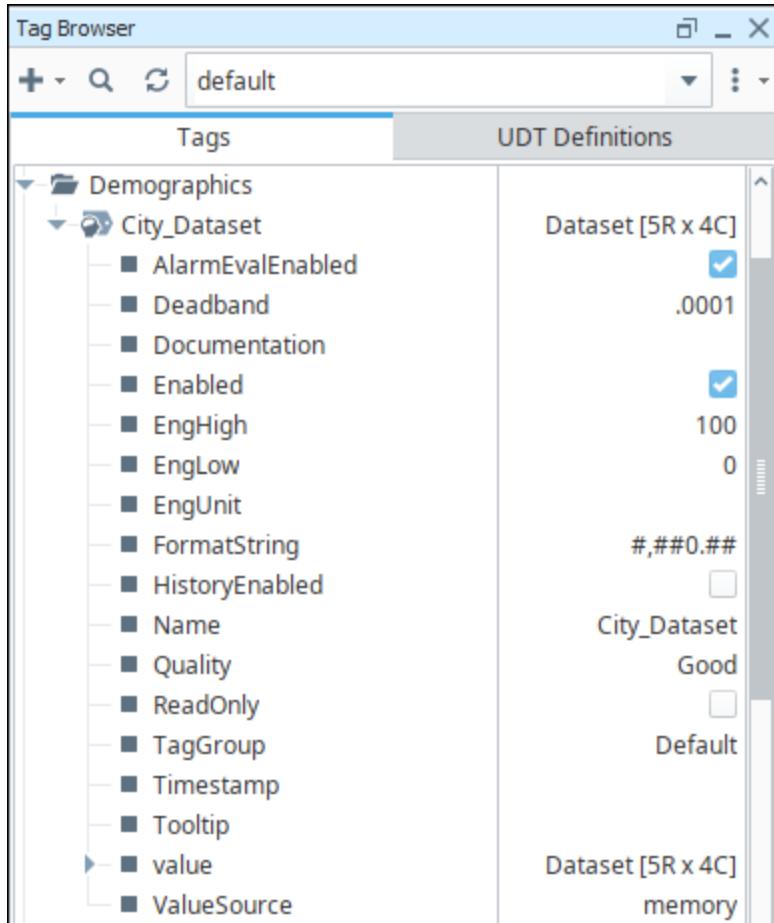
Many OPC servers and drivers already support array type Tags, and now each element in the array can easily be represented with the array data types in Ignition. Additionally, array data types can be used with devices that do not support array types, and will instead expose each bit in the value. Because the core data type of each element in the array is the same, it is possible to apply Tag History, Alarming, or Scaling configurations onto the array, and these configurations will be inherited by each element.

Array Tag Write-Back

OPC Array Tags support writing back to the device. How this is done can vary, depending on the type of OPC Server in use. Some OPC Servers support writes to individual array elements, where a write would occur just like any other Tag write. However, some OPC Servers do not support individual element writes, which means the whole array will need to be written back to the array Tag, even if only a single element is changing.

Dataset Tags

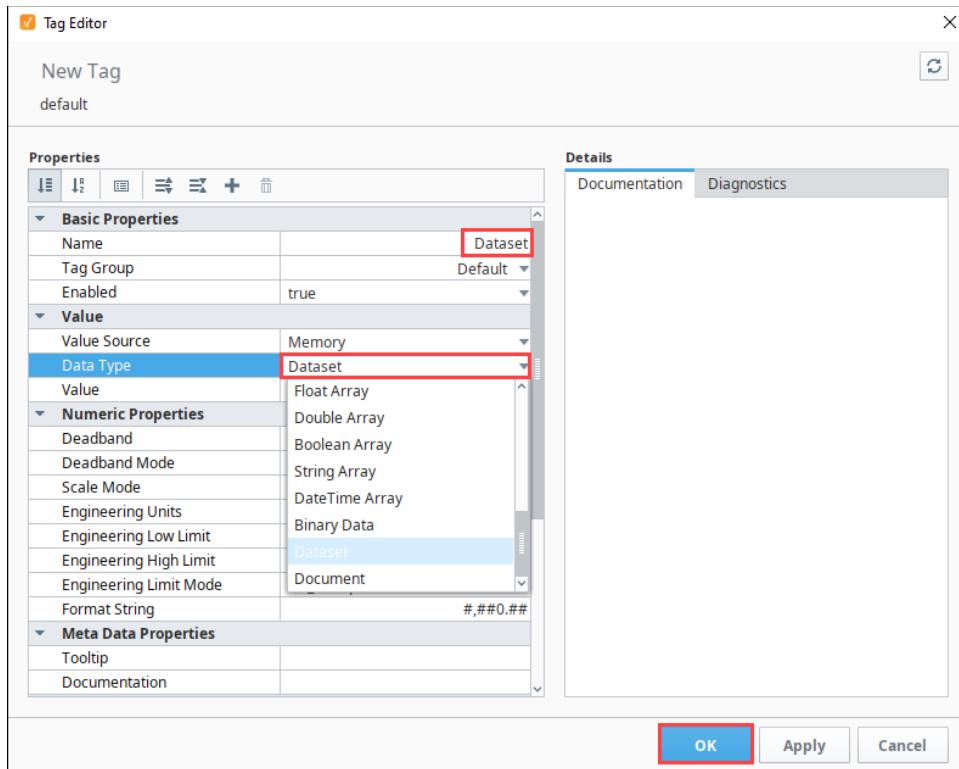
Dataset Tags allow multiple rows and columns worth of data to be stored in a Tag. Each column is exposed as a separate folder in the Tag (i.e., the "name" folder in the image below). Dataset Tags can be driven by a query, so it's possible to query for multiple columns on a row in a single Tag. This is much more efficient than using multiple query Tags (and thus multiple queries) to retrieve the same data.



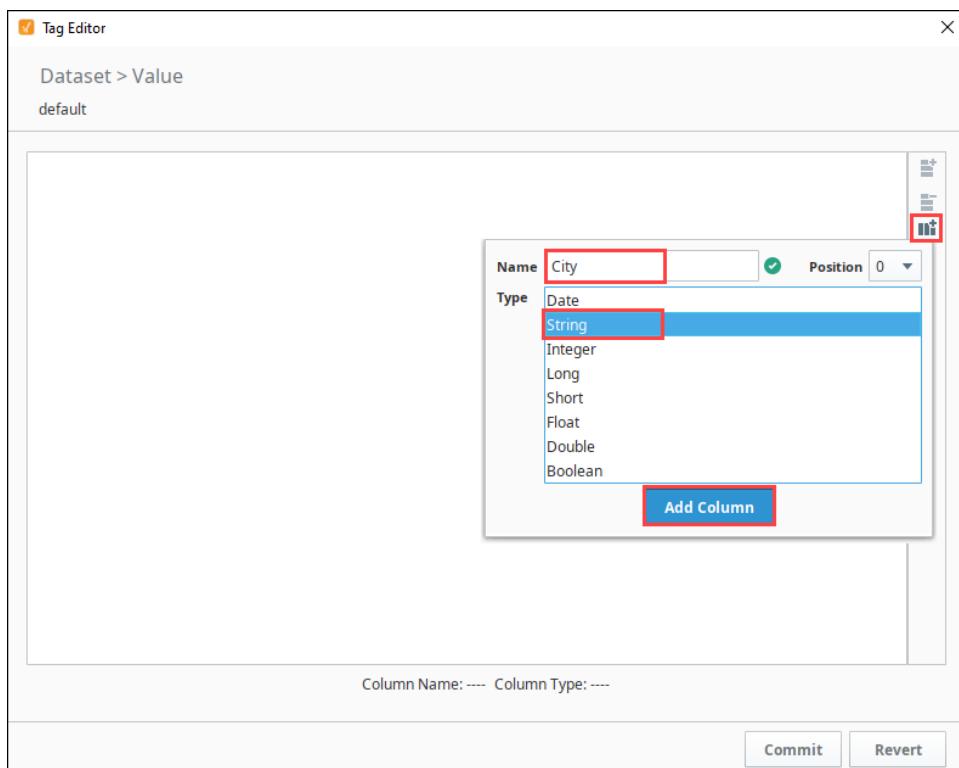
Dataset Tag Example

The following example will create a dataset memory Tag and display the contents in a Table component.

1. Create a new **Memory Tag**. Name it **Dataset**, and change the data type to **Dataset**. The Dataset will be empty by default.



2. Click the **Edit** icon next to **Value**. The Value screen is displayed. For this example, we created a simple dataset with four columns and five rows.
3. Click the **Add Column** icon. Name the first column **City** and set type to be **String**.
4. Click **Add Column**.



5. Repeat adding columns as follows:

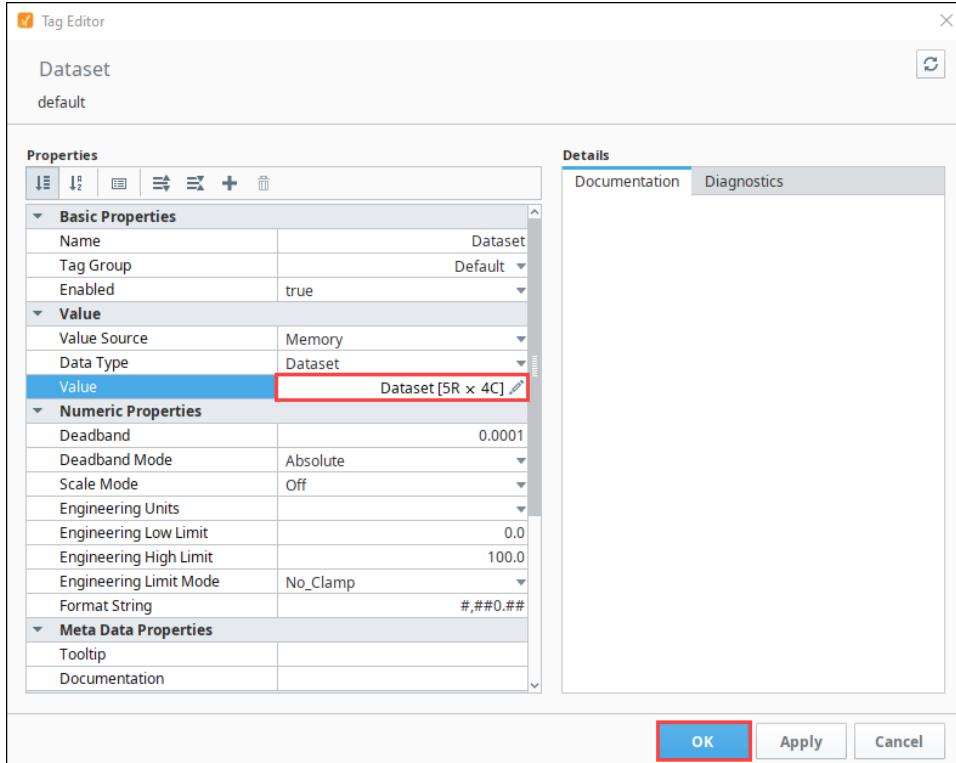
Column Name: **Population** Type: **Integer**
 Column Name: **TimeZone** Type: **String**

Column Name: **GMTOffset** Type: **Integer**

6. Click the **Add Row**  icon. Add the row information as follows:

New York	8368710	EST	-5
Los Angeles	3833995	PST	-8
Chicago	2853114	CST	-6
Houston	2242193	CST	-6
Phoenix	1567924	MST	-7

7. Click the **Commit** button.
8. Click **OK** to save the Tag.



The Tag will now contain rows, columns, and values based on the configurations you made earlier in this example. Now you have a Tag with a dataset value that can be bound to by components in Vision and Perspective.

Tag Paths

Tags and their properties can be referenced by a string-based path in many areas of Ignition, such as [expressions](#) and [scripts](#). Each Tag has a unique absolute path and often has many equivalent relative paths when referenced from other Tags. In most cases these paths are generated automatically via helper buttons. However, it's a good idea to understand how Tag paths work, particularly if you need to configure an Indirect Tag Binding, or access a Tag from an expression, or script.

A Tag path looks something like this: [Tag Provider]folder/path/tag.property

The folder/path/tag.property portion of the path may contain the following:

- A Tag
- Any number of nested folders followed by a Tag, separated by forward slashes (/)
- A period (.) followed by a property name after the Tag. Omitting this is equivalent to using the . value property

The [Tag Provider] portion surrounded by square braces can have the following options:

Source Option	Meaning
[Tag Provider Name]	The name of the Tag provider that hosts the Tag.
[] or not specified	The default Tag provider for the current project. If used in the Gateway scope, this notation can (generally) result in an invalid path, as the Gateway doesn't have a default Tag provider.
[.]	Relative to the folder of the Tag that is being bound. This is especially useful in UDT definitions.
[~]	Relative to the Tag provider of the Tag that is being bound (root node).
[Client]	Refers to the Vision Client Tag provider, which contains only Vision Client Tags.
[System]	Refers to a System Tag.

On this page ...

- [Using Relative Tag Paths](#)
- [Tag Path Manipulation](#)
- [Array Type Tag Paths](#)
- [Document Type Tag Paths](#)
 - [Writing to Document Type tags](#)
- [Using the {this} Keyword](#)
 - [Using {this} in Alarms](#)

Using Relative Tag Paths

Tag paths that begin with [.] or [~] are known as *relative paths*. They are used inside Tags that bind to other Tags, and they are relative to the host Tag's path. Using the relative path syntax helps to avoid problems caused by moving Tags and renaming providers.

[.] refers to the Tag's current folder. By using [.], Tags can be moved from folder to folder without problem (provided that all of the applicable Tags are moved together). Additionally, you can use ".." (two periods) to go back one folder from the current relative position, for example [..]/../tag allows you to reference a Tag that is two folders up.

[~] refers to the Tag's provider root. It replaces an explicit provider name and thus protects the Tag path from "breaking" if the provider is renamed or if the Tag is imported/exported/moved between different providers.

Tag Path Manipulation

Ignition provides a great deal of flexibility for Tag addressing since Tag paths and Tag properties are string-based. The underlying strings that compose a valid Tag path can be assembled from many different parts in which the eventual construction results in a valid Tag path.

The following scripting demonstrates this concept. Suppose there was a Tag path to a level indicator in a tank. In this case it is the default Tag provider, Tanks folder, Tank 1 Folder, and the Level Tag.

```
tagPath = "[default]Tanks/Tank 1/Level"
```

But suppose that there was more than just Tank 1 and instead there was Tank 2, Tank 3, Tank 4, etc. Dynamically changing the Tag paths is simple because Ignition's Tag paths are string representations. The following takes the tank number and inserts it into a new Tag path. The tankNumber variable changes the eventual creation of the tagPath. Using this method in scripting or in an expression binding will look slightly different.

Python Dynamic Tag Path

```
tankNumber = 2  
tagPath = "[default]Tanks/Tank %i/Level" % tankNumber
```

Expression Dynamic Tag Path

```
tag( "[default]Tanks/Tank "+{Root Container.tankNumber}+"/Level").value
```

The result of the tagPath variable will be **[default]Tanks/Tank 2/Level** which is a valid Tag path to the level sensor for Tank 2.

Array Type Tag Paths

When a path leads to an array type tag, individual elements can be accessed using square brackets, and the index offset.

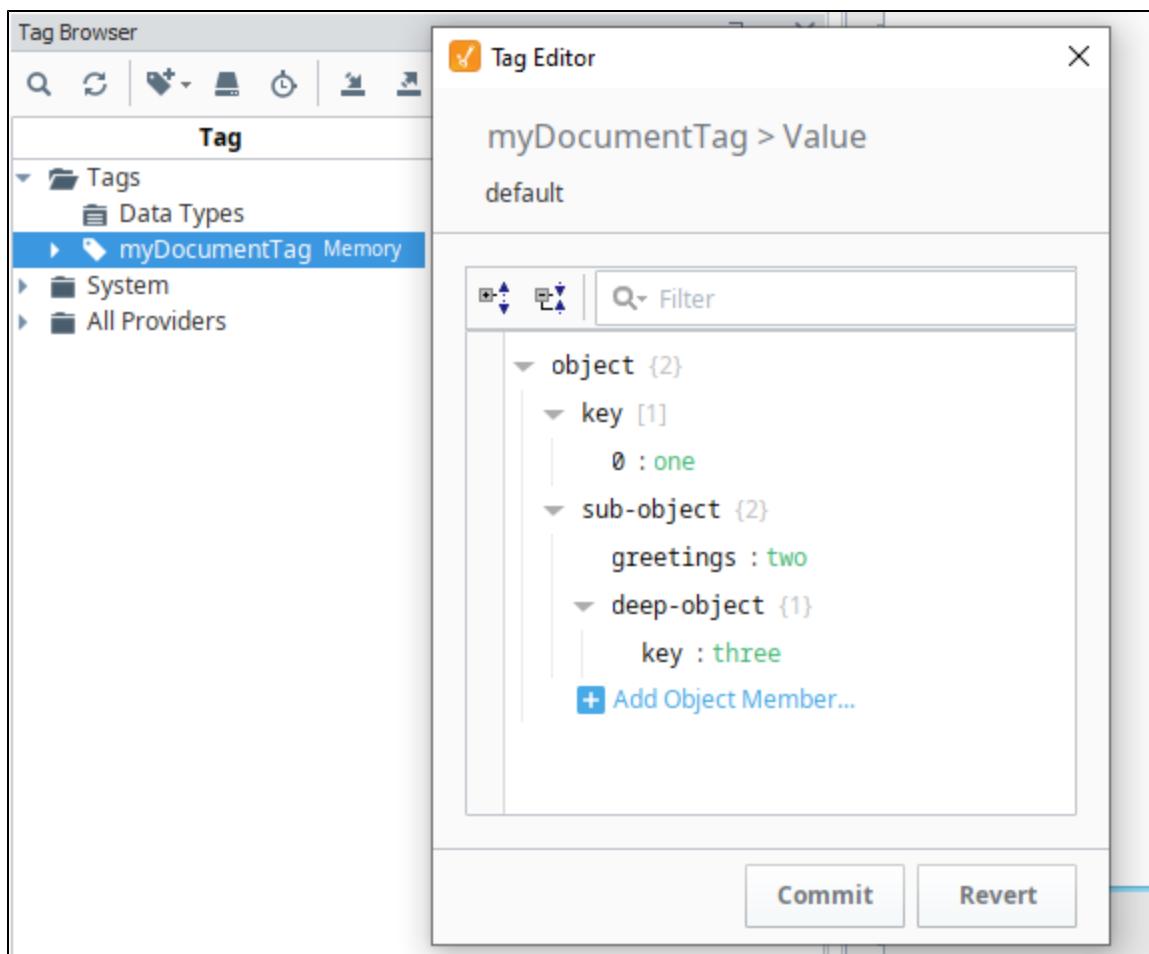
```
[default]Folder/myArrayTag[0]
```

Document Type Tag Paths

This feature is new in Ignition version **8.1.0**
[Click here](#) to check out the other new features

When a path leads to a document type tag, individual objects within the tag's value can be accessed by using square brackets wrapped around a set of quotations marks that lead to the object.

For example, say we have a documentation tag with the following value:



The following would return "one", which is in the first element of the "key" array:

```
[default]myDocumentTag['key[0]']
```

JSON Values are references with a '.' character, so the following would retrieve the value of "two" inside of the "sub-object":

```
[default]myDocumentTag['sub-object.greetings']
```

For more complex structures, you can continue adding to the JSON string. The following would return "three" from the key in "deep-object":

```
[default]myDocumentTag['sub-object.deep-object.key']
```

Writing to Document Type tags

The paths above aren't only for reading. Using the same addressing mentioned above on a bi-directional component binding would allow the bidirectional binding to write back to the specific JSON object in the document.

Using the {this} Keyword

Tags have a built-in "this" keyword, that can be used as a reference to the Tag. The keyword is useful in cases where an expression is being configured on a Tag property and you want to reference the value of another property.

```
// The expression below always returns the name of the Tag
{this.name}
```

Using {this} in Alarms

The "this" keyword is also available for use on expressions on alarms. However when used like this, the keyword still refers to the Tag, not the alarm. Thus, "`this.name`" on an alarm property expression would return the name of the Tag, not the name of the alarm.

For more information on Ignition's Expression language, see [Expression Overview and Syntax](#).

Related Topics ...

- [Tag Event Scripts](#)

Tag Quality and Overlays

Tag Quality

Ignition has Quality built into Tags automatically. Data Quality is the measure of the reliability of a particular Tag's data. If a Tag's quality is not Good, the value generally should *not* be trusted. In these cases an overlay will be shown on bound components. In addition, the Tag in the Tag Browser will also be decorated with a quality icon, signifying a non-good value.

Tag Quality in the Designer

In the Tag Browser, find your Tag, expand it, and scroll down to the meta property called **Quality**. Here, you can check the quality of the Tag. This example shows a Good Quality Tag, meaning the Tag can be trusted.

Tags		UDT Definitions	
Tank Level		96	Double
■ AlarmEvalEnabled		<input checked="" type="checkbox"/>	Boolean
■ Deadband		.0001	Double
■ Documentation			String
■ Enabled		<input checked="" type="checkbox"/>	Boolean
■ EngHigh		100	Double
■ EngLow		0	Double
■ EngUnit			String
■ FormatString		#,##0.##	String
■ HistoricalDeadband		null	Double
■ HistoryEnabled		<input checked="" type="checkbox"/>	Boolean
■ HistoryMaxAge		null	Integer
■ HistoryProvider		vm_db	String
■ HistoryTimeDeadband		null	Integer
■ Name	Tank Level		String
■ Quality	Good		String
■ ReadOnly			Boolean

One obvious indicator if the Tag is of bad quality is if there is a red error message next to the Tag in the Value column. Hover over the error message to see if there is any additional information about the error. You can also expand the Tag to see the quality issue. This example shows the Query Tag with a Error Expression Evaluation which helps you resolve the issue promptly.

Tags		UDT Definitions	
Query Tag	Error_ExpressionEval	Integer	Boolean
■ AlarmEvalEnabled	Error_ExpressionEval	<input checked="" type="checkbox"/>	Boolean
■ Datasource		.0001	Double
■ Deadband			String
■ Documentation		<input checked="" type="checkbox"/>	Boolean
■ Enabled		100	Double
■ EngHigh		0	Double
■ EngLow			String
■ EngUnit		#,##0.##	String
■ FormatString			Boolean
■ HistoricalDeadband		Query Tag	String
■ HistoryEnabled			String
■ HistoryMaxAge			Boolean
■ HistoryProvider			String
■ HistoryTimeDeadband			String
■ Name	Error_ExpressionE...		DateTime
■ Quality			String
■ Query			String
■ ReadOnly			Boolean
■ TagGroup			String
■ Timestamp			DateTime
■ Tooltip			String
■ value	Error_ExpressionEval	db	Integer
■ ValueSource			String

On this page ...

- Tag Quality
 - Tag Quality in the Designer
- Component Overlays
 - Perspective Component Overlays
- Quality Code Reference Table
 - Vision Component Overlays
 - Vision Quality Code Reference Table
- Tag Quality and Referenced Tags
- Overlay Opt-Out



Tag Quality and Overlays

[Watch the Video](#)

Component Overlays

It is especially important in HMI screens to be able to gauge the health and accuracy of what is displayed at a glance. In a highly distributed system like Ignition, it is especially important as the client may be located at quite a distance (maybe across the world) from the physical process it is monitoring and controlling.

For these reasons, Perspective and Vision components display visual overlays for various reasons to indicate that the data they are displaying is not good, or pending a reply from the device. Each data binding that drives a component is evaluated for quality. If any of these qualities becomes poor, the Perspective or Vision component will show an overlay. The different overlays can mean different things, denoting their underlying cause. What they indicate is based on the Quality properties of Tags.

Component overlays appear in the Designer workspace, Perspective Session, and Vision Client to let designers and operators know when there is a problem with one of the bindings on a component. What is cool about component overlays is that they not only tell you that there is a problem, but they also help diagnose the problem. Vision and Perspective overlay systems are similar, but each look a little bit different.

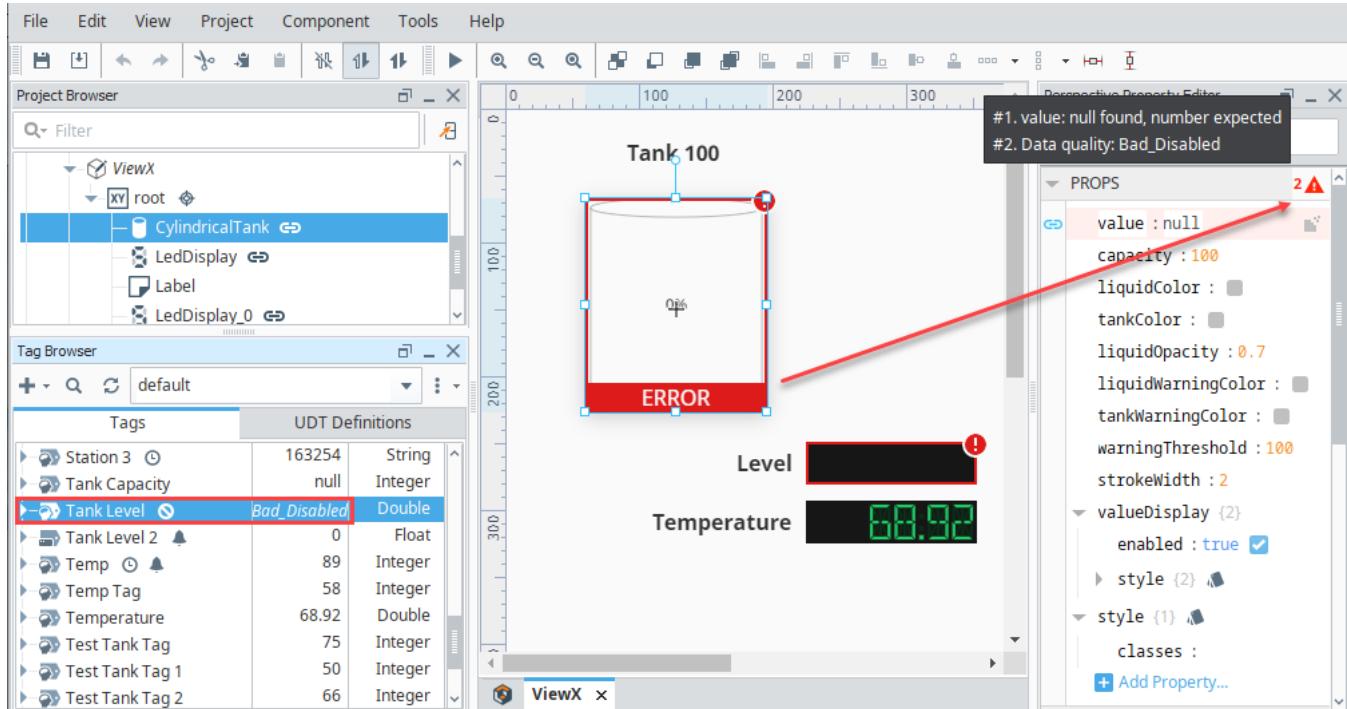
The sections below describe in detail Perspective and Vision overlays. Each module has its own Tag Quality Code Reference Table displaying the error codes and what they mean.

Perspective Component Overlays

Perspective overlays give designers and operators a lot of information to help them diagnose and correct a problem. Component overlays look and behave slightly different in Designer Mode, Preview Mode, and in a Perspective Session. Each is described below.

Component Overlays in Designer Mode

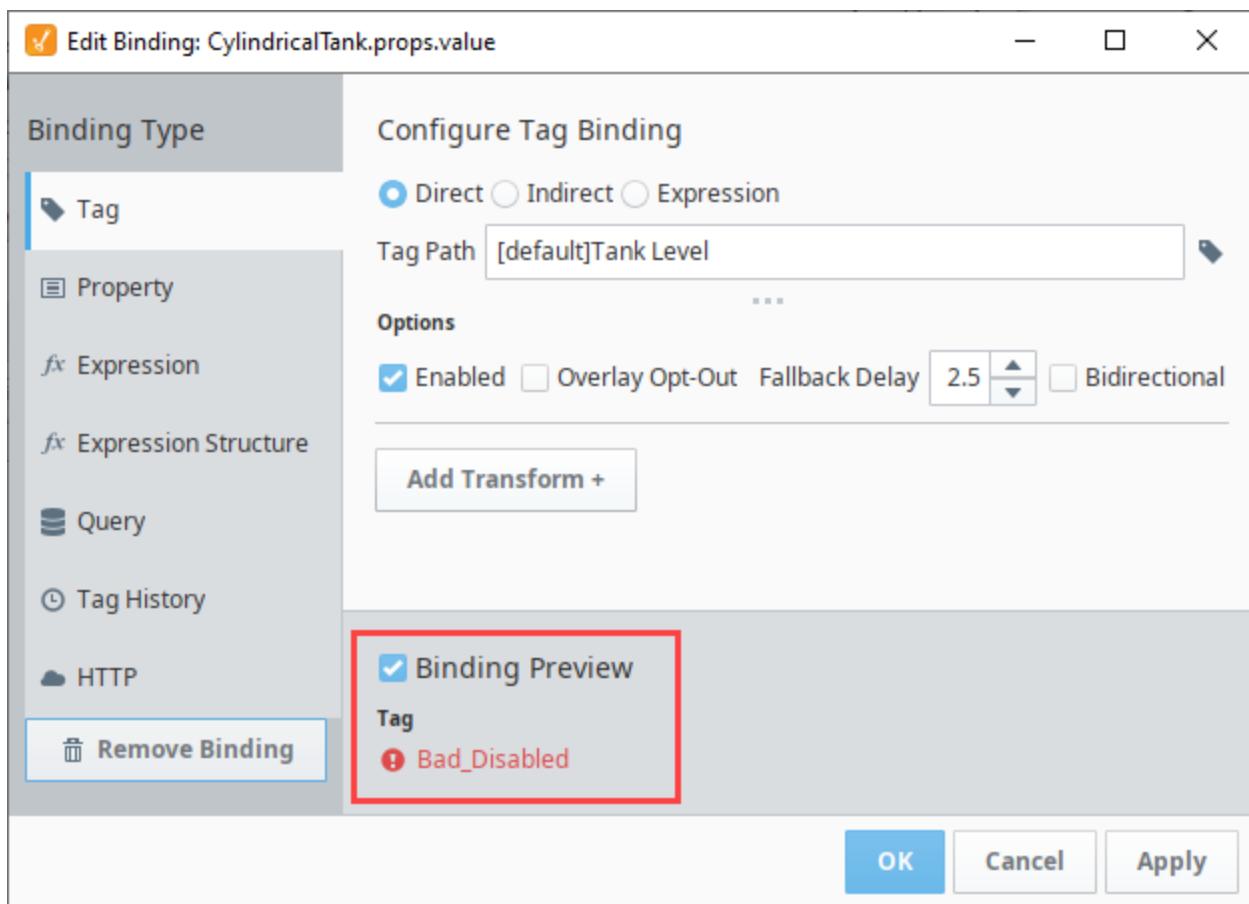
In the following example, you see the Cylindrical Tank and a LED Display component have overlays with a red border around them and an  icon. If you select the component, a red triangle will appear in upper right corner of the Property Editor giving the designer some information about the error. Hover over the red triangle to open the message box to identify the data quality error and cause. You'll also notice that the affected property (i.e., value) is highlighted in light red indicating this property is the one having a problem. If the error is binding related, click on the binding icon next to the property to open the Edit Binding window.



Under the Binding Preview area, you can see there is a Configuration Error. If the problem is with a binding type, chances are you might be able to correct it here, or you may need to go directly to the Tag, or check other resources like a database, OPC server, etc.

Component Overlays

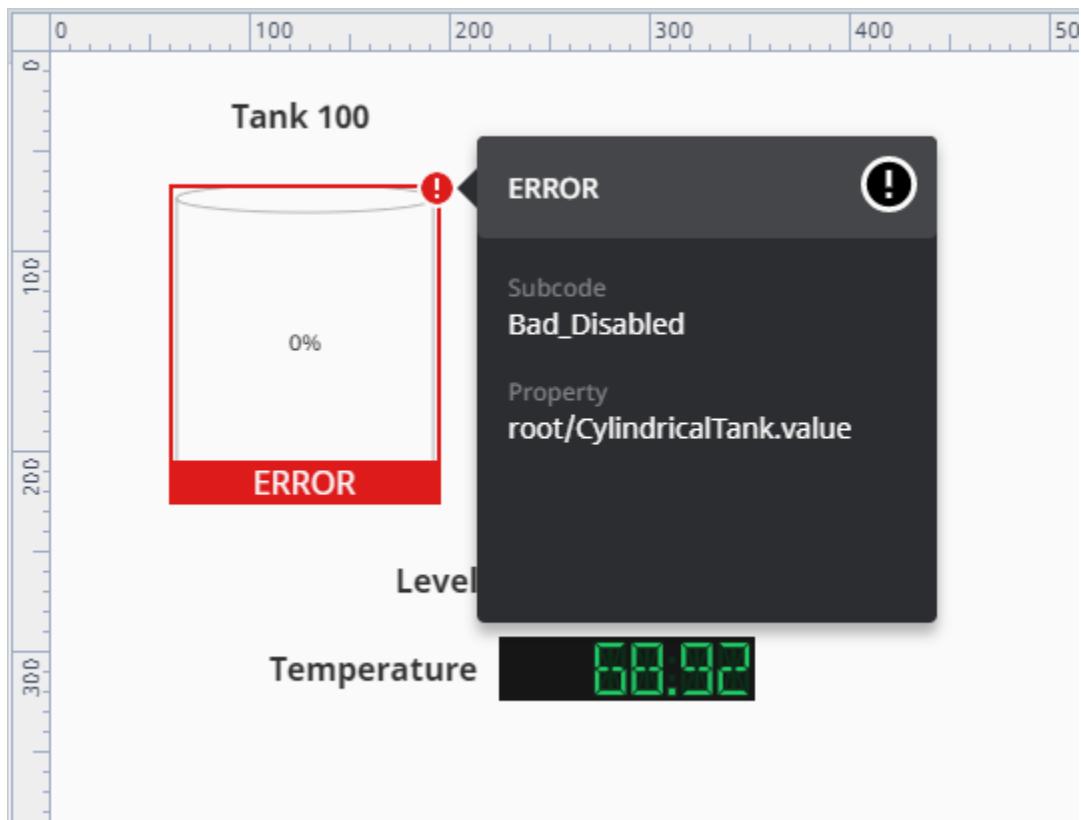
[Watch the Video](#)



Component Overlays in Preview Mode

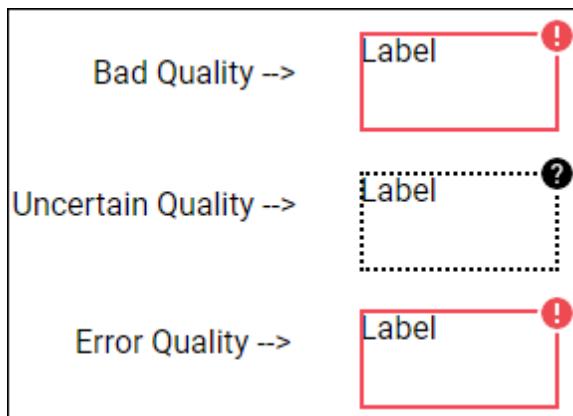
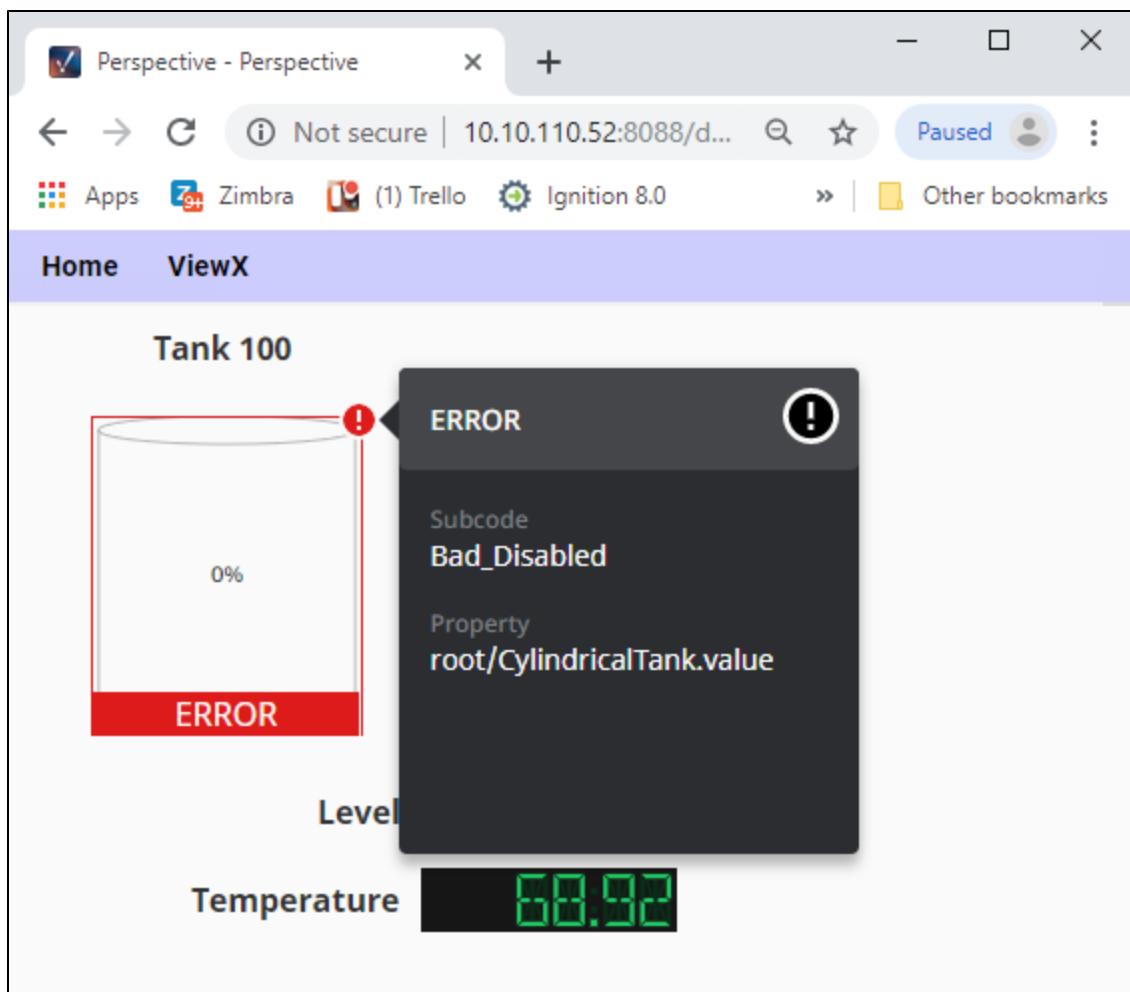
Switch from the Designer mode to the Preview Mode. To put your active view in **Preview Mode**, press the Preview / Designer mode ► icon in the top menubar. Components that have a problem will have a red overlay and exclamation mark around them. To see the error, click on the exclamation

mark  . This opens a message box that informs you that you have an error and identifies affected property.



Component Overlays in a Perspective Session

Component overlays in a Perspective Session work the same way as they do in Preview Mode. The component with the problem will have a red overlay with an exclamation mark. Click on the exclamation mark to open the message box. It will identify the data quality error and affected property to help to identify the problem. To correct the problem, you'll have to go back to the Designer.



Quality Code Reference Table

There are four primary quality codes which quickly inform the user of the quality of the Tag: Good, Uncertain, Bad, and Error.

Each quality code has a range of subcodes that provide more specific information about the value of the Tag. The following tables outline the primary data qualities. Quality codes are broken up into several ranges. Note that not every value within a range has a code: the space is there for future codes to be added.

- 0-225: Good quality. The value is generally considered reliable.
- 256 - 511 : Uncertain quality. Generally represents a value that was good, but the reliability is somewhat questionable. These are generally used when the system hasn't received a new value in a timely manner.
- 512 - 767: Bad quality. There's a problem with the value, but it's an "expected" or well recognized type of the problem is. For example, the trial expired or read access was explicitly denied.

- 768 - 1023: Error quality. There is a problem with the value, and the problem was completely unexpected. For example, a query on a query tag couldn't execute, but the Tag doesn't know why. More details on such errors are usually sent to a console somewhere, such as the Gateway's console.

Good Quality	Subcodes	Meaning
Good_Unspecified	0	A generic "good" code. Generally used in conjunction with a matching good quality subcode, (1,2, or 192).
Good_Provisional	1	Good data that should not be considered valid over long periods of time. Provisional values will not cache.
Good_WritePending	2	Used when a write is in progress. Generally, values use this code until the system knows the write through successfully, which would then result in a 192 code.
Good	192	This data has met all criteria for being considered reliable.
Good_Overload	202	<p>This feature is new in Ignition version 8.1.2 Click here to check out the other new features</p> <p>Represents good data that is being sampled slower than requested due to a resource limitation.</p>
Uncertain Quality	Subcodes 256 - 261	Meaning
Uncertain	256	An unspecified degree of uncertainty exists in this value.
Uncertain_LastKnownValue	257	The current value is unavailable and represents the last known value.
Uncertain_InitialValue	258	Indicates that a subscription has been made and a good value should be arriving shortly.
Uncertain_DataSubNormal	259	Insufficient good-quality sources required for the derivation of this value.
Uncertain_EngineeringUnitsExceeded	260	Indicates that a value has gone beyond its configured engineering units.
Uncertain_IncompleteOperation	261	An async operation is currently pending and its result is unknown.
Bad Quality	Subcodes 512 - 767	Meaning
Bad	512	General quality for a bad quality.
Bad_Unauthorized	513	An unauthorized request was made for data that requires authorization.
Bad_AccessDenied	514	Data requested that requires credentials not held by the requesting user.
Bad_Disabled	515	Data source is currently not enabled.
Bad_Stale	516	Data is out-of-date based upon the requested refresh interval.
Bad_TrialExpired	517	The Trial Mode's timer expired.
Bad_LicenseExceeded	518	The license limit has been exceeded.
Bad_NotFound	519	Object requested was not found.
Bad_ReferenceNotFound	520	Derived or referenced value required an object which was not found.
Bad_AggregateNotFound	521	Requested aggregate was not found.
Bad_NotConnected	522	A connection required for this value is not currently connected.

Bad_Gateway CommOff	523	Connection to the Ignition Gateway is currently turned off.
Bad_OutofRange	524	This value exceeded its allowed range.
Bad_Database NotConnected	525	A database connection required for this value is not connected.
Bad_ReadOnly	526	The target is not writable / read only.
Bad_Failure	527	A "failure" code was received from the underlying system. Additional details may be in the diagnostic message. This generally does not indicate an exception, which would be handled by Error_Exception, but instead a simple failure from a system that can return success or failure.
Bad_Unsupported	528	The operation is not supported by the target.
Error Quality	Subcodes 768 - 1023	Meaning
Error	768	An unexpected error occurred while retrieving or calculating this value.
Error_Configuration	769	The source of this value is not configured correctly.
Error_ExpressionEval	770	The source expression was unable to be executed.
Error_TagExecution	771	The source Tag could not be executed.
Error_TypeConversion	772	The actual value was not able to be coerced into the configured data type for the source of this value.
Error_DatabaseQuery	773	A database query required for this value caused an error upon execution.
Error_IO	774	An input/output error occurred while attempting to retrieve or calculate this value.
Error_TimeoutExpired	775	An async operation failed due to a timeout.
Error_Exception	776	An exception was caught, and logged in the relevant system.
Error_InvalidPathSyntax	777	A path (i.e., Tag path, property path, etc.,) was not able to be parsed because the syntax is invalid.
Error_Formatting	778	Attempted formatting (i.e., numeric, date formatting) failed.
Error_ScriptEval	779	A script needed to create this value failed to execute.
Error_CycleDetected	780	Calculating the value involved an execution cycle.

Vision Component Overlay Chart

For legacy reasons, Vision has some quality codes that are different from the quality codes listed above. Specifically, Vision receives the codes above, but also performs some translations, mapping to the entries listed below. E

Quality	Value	Meaning
OPC_Bad_Data	0	The data is not reliable, further data isn't available.
OPC_Config_Error	4	There is a problem with the OPC Server, or OPC Item Path on the Tag.
OPC_Not_Connected	8	A connection required for this value is not currently connected.
OPC_Device_Failure	12	There was an issue with the device connection, making the value unreliable.
OPC_Sensor_Failure	16	There has been a failure in the sensor from which the value is derived.
OPC_Bad_Showing_Last	20	Received a bad quality from the OPC server. Also implies that the value represents the last known good quality value.
OPC_Comm_Failure	24	Received a "communication failure" message from the OPC server.

OPC_Out_Of_Service	28	The OPC server is reporting that the source of the data is not operational.
OPC_Waiting	32	Waiting for the server to obtain values.
OPC_Uncertain	64	The server identifies the value as uncertain, but is unaware of the reason.
OPC_Uncertain_Showing_Last	68	Whatever was updating this value has stopped doing so. The value represents the last known good value.
OPC_Sensor_Bad	80	There has been a failure in the sensor from which the value is derived by the data source.
OPC_Limit_Exceeded	84	The value is outside of the range of values defined for this parameter.
OPC_Sub_Normal	88	The value is derived from multiple sources, and has less than the required number of Good sources.
Good	192	The data has met all criteria for being considered reliable.
Unknown	256	The Tag has an unknown value, and a reason has not been provided.
Config_Error	300	There is a problem with the Tag's configuration. The error log may provide more information as to the exact problem.
Comm_Error	301	There is a problem in communication somewhere between the Tag and its data source.
Expression_Eval_Error	310	The expression in the Tag generated an error during execution. The error log should provide more information on the error.
SQL_Query_Error	311	A database connection is required for this value, but there is an error with the SQL syntax driving the value.
DB_Connection_Error	312	A database connection is required for this value, and the connection is reporting some sort of error.
Good (Provisional)	320	Temporary "Good" value. The value isn't cached because the underlying quality may differ than what appears on the Tag.
Tag_Exec_Error	330	There was an error when evaluating the Tag.
Type_Conversion_Error	340	The value of the Tag could not be converted to the requested data type. Check the assigned data type of the Tag.
Access_Denied	403	The Tag permission settings do not allow the current user to view the Tag.
Not_Found	404	The Tag, or a Tag referenced from inside of it, could not be found (incorrect reference path).
Disabled	410	The Tag's "enabled" property has been set to false.
Stale	500	The Tag has not been evaluated within the expected time frame. There is likely a deeper problem with the Tag provider.
Unknown	600	An unspecified degree of uncertainty exists in this value.
Write_Pending	700	Used when a write is in progress. Generally, values use this code until the system knows the write through successfully, which would then result in a 192 code.
Demo_Expired	900	The system driving the Tag is operating in demo mode and has timed out.
GW_Comm_Off	901	When viewing Tags in the Designer, the Tags will have this value if communication with the Gateway is turned off from the toolbar.

Bad Quality OPC_BAD_DATA (0)		Config Error OPC_CONFIG_ERROR (4)	
Not Connected OPC_NOT_CONNECTED (8)		Device Failure OPC_DEVICE_FAILURE (12)	
Sensor Failure OPC_SENSOR_FAILURE (16)		Bad, showing last value OPC_BAD_SHOWING_LAST (20)	
Comm. Failure OPC_COMM_FAIL (24)		Out of Service OPC_OUT_OF_SERVICE (28)	
Waiting OPC_WAITING (32)		Uncertain OPC_UNCERTAIN (64)	
Uncertain, showing last value OPC_UNCERTAIN_SHOWING_LAST (68)		Bad Sensor OPC_SENSOR_BAD (80)	
Limit Exceeded OPC_LIMIT_EXCEEDED (84)		Sub-normal OPC_SUB_NORMAL (88)	
Unknown OPC_UNKNOWN (256)		Config Error CONFIG_ERROR (300)	
Comm. Error COMM_ERROR (301)		Evaluation Error EXPRESSION_EVAL_ERROR (310)	
SQL Error SQL_QUERY_ERROR (311)		DB Connection Error DB_CONN_ERROR (312)	
Tag Evaluation Error TAG_EXEC_ERROR (330)		Type Conversion Error TYPE_CONVERSION_ERROR (340)	
Access Denied ACCESS_DENIED (403)		Not Found NOT_FOUND (404)	
Disabled DISABLED (410)		Stale STALE (500)	
Unknown UNKNOWN (600)		Write Pending WRITE_PENDING (700)	
Demo Expired DEMO_EXPIRED (900)	 TRIAL EXPIRED TRIAL EXPIRED TRIAL EXPIRED	Gateway Comm. Off GW_COMM_OFF (901)	

Vision Component Overlays

An overlay on a Vision component lets the operator know that they could be looking at a bad value for that Tag. When the overlay goes away and the values start coming in again, the operator knows that it's a valid Tag, and the values can be trusted.

Component Overlays in Designer Mode

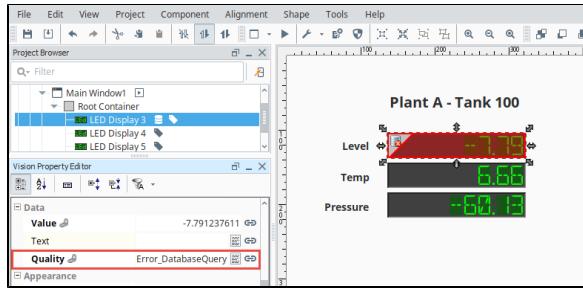
In the following example, you see a red overlay with an icon in the top left corner of the selected LED Display component. The icon gives you a clue to the source of the problem. In this example, it is an SQL Database error. In the Vision Property Editor, the Quality property is highlighted and you'll notice there is a "Error_DatabaseQuery" error message.

The overlays table in the next section show all the possible Vision overlays and what they mean.



Tag and Component Overlays

[Watch the Video](#)

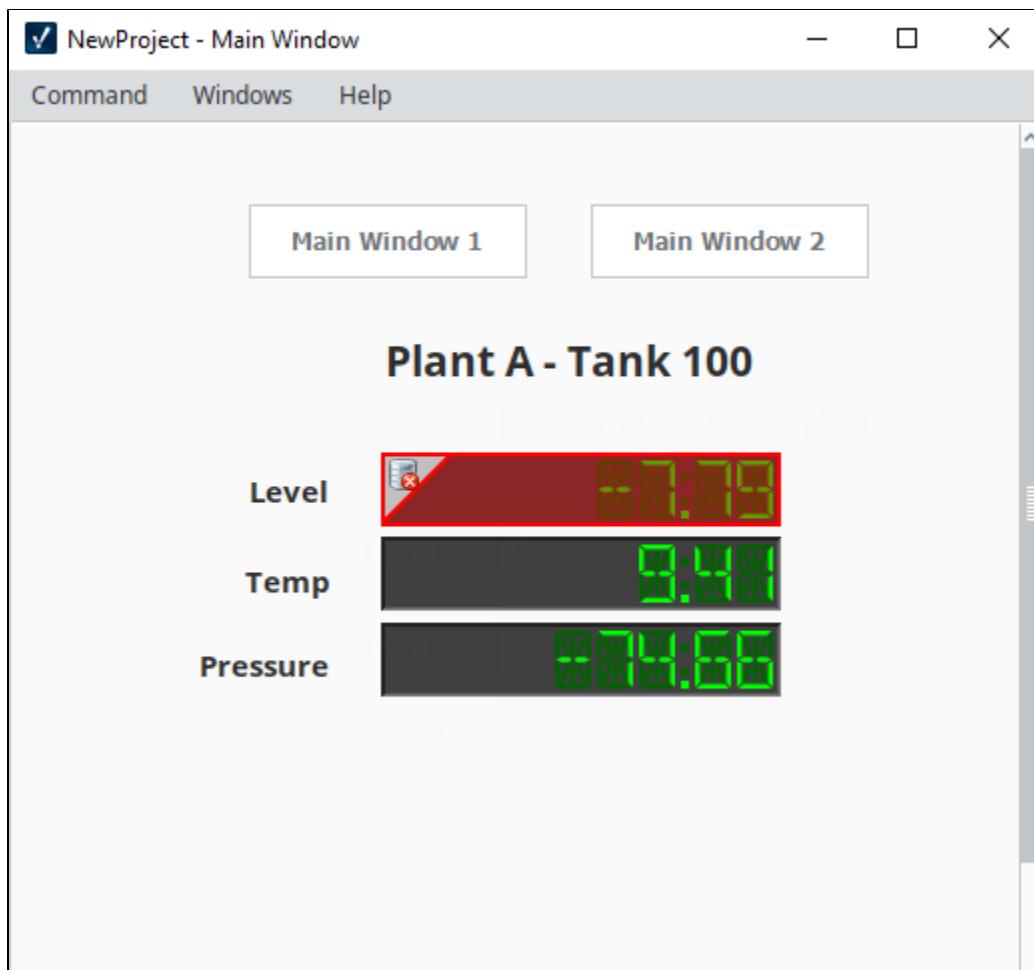


Component Overlays in Preview Mode

Let's switch from the Designer mode to the Preview Mode. To put your active view in **Preview Mode**, press the Preview / Designer mode icon ► in the top menubar. Components that have a problem will have a red overlay and an icon in the top left of the component overlay to indicate the problem. The overlay is identical to the overlay that is displayed in the Designer, but the component cannot be selected.

Component Overlays in the Vision Client

Component overlays in a Vision Client work the same way as they do in Preview Mode of the Designer. You have to look at the icon on the overlay to help you diagnose the problem. Go back to the Designer to correct the problem.



Vision Quality Code Reference Table

The following table outlines the primary data qualities. The most important is **Good**, and that has a value of **192**. There are more values, but these represent the most common:

Quality	Value	Meaning
Type_Conversion_E	340	The value of the Tag could not be converted to the requested data type. Check the assigned data type of the Tag.

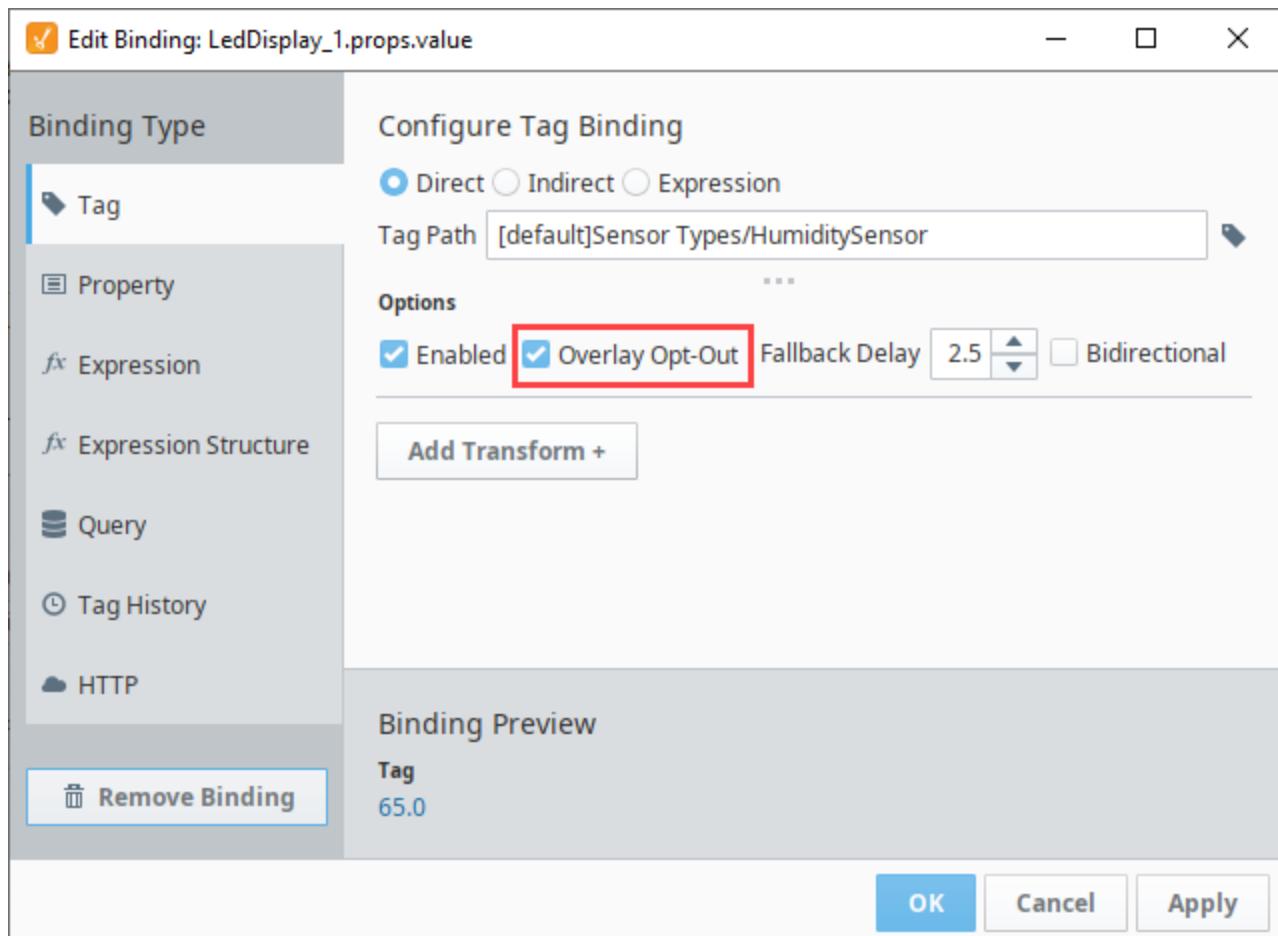
rror		
Tag_Exec_Error	330	There was an error when evaluating the Tag.
Stale	500	The Tag has not been evaluated within the expected time frame. There is likely a deeper problem with the Tag provider.
OPC_Not_Connected	8	The OPC server driving the Tag is not currently connected OR a value has not yet been received by the Tag from the server.
OPC_Bad_Data	0	The data is not reliable, further data isn't available.
Not_Found	404	The Tag, or a Tag referenced from inside of it, could not be found (incorrect reference path).
GW_Comm_Off	901	When viewing Tags in the Designer, the Tags will have this value if communication with the Gateway is turned off from the toolbar.
Good (Provisional)	320	Temporary "Good" value. The value isn't cached because the underlying quality may differ than what appears on the Tag.
Good	192	The data has met all criteria for being considered reliable.
Expression_Eval_Error	310	The expression in the Tag generated an error during execution. The error log should provide more information on the error.
Driver_Demo_Timeout	900	The system driving the Tag is operating in demo mode and has timed out.
Disabled	410	The Tag's "enabled" property has been set to false.
Config_Error	300	There is a problem with the Tag's configuration. The error log may provide more information as to the exact problem.
Comm_Error	301	There is a problem in communication somewhere between the Tag and its data source.
Access_Denied	403	The Tag permission settings do not allow the current user to view the Tag.

Tag Quality and Referenced Tags

When Tags reference other Tags, such as in expressions, they will often pass the worst sub-quality up as their own. For example, even though a particular Tag's expression executes without problem, if the expression references a Tag whose quality is "Bad", the expression Tag will also report "Bad."

Overlay Opt-Out

Choosing the Overlay Opt-Out option will ignore the quality of the chosen Tag, making it have no effect on the component's quality overlay. The Overlay Opt-Out option is located in the Tag bindings for both Perspective and Vision components. If this option is enabled, the operator will not see any overlays and will have no indication that the underlying Tag quality is something other than good. A word of caution when you use the Opt-Out option because you always want to give the operator some indication that the values they are seeing on the screen can be trusted, and by opting out, you are removing that indicator for the operator.



Related Topics ...

- [Tag Scaling Properties](#)
- [Tag Properties](#)
- [Bindings in Perspective](#)
- [Tag Bindings in Vision](#)
- [Indirect Tag Bindings in Vision](#)

Exporting and Importing Tags

Ignition can export and import Tag configurations to and from the **JSON** (JavaScript Object Notation) format. You can import **XML** (Extensible Markup Language) or **CSV** (Comma Separated Value) file formats as well, but Ignition will convert them to **JSON** format while in a tag provider. Tag export files can be edited directly in any text editor, allowing you to make bulk edits to tags before importing them back into a Tag provider.

Export Tags

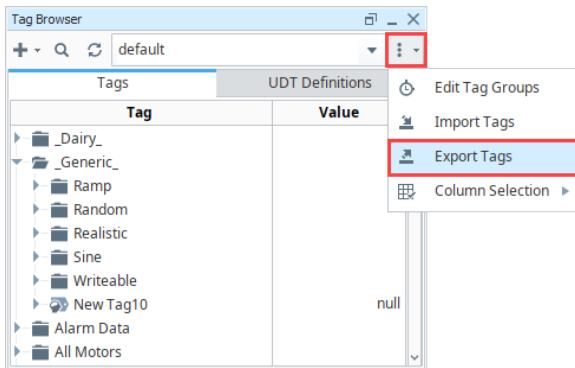
1. In the **Designer**, go to the **Tag Browser**, and select the Tags tab to export all your Tags. You can also select an individual folder that contains Tags you want to export. You can even export individual Tags as long the individual Tags are in the same folder.

Exporting UDTs and UDT Instances

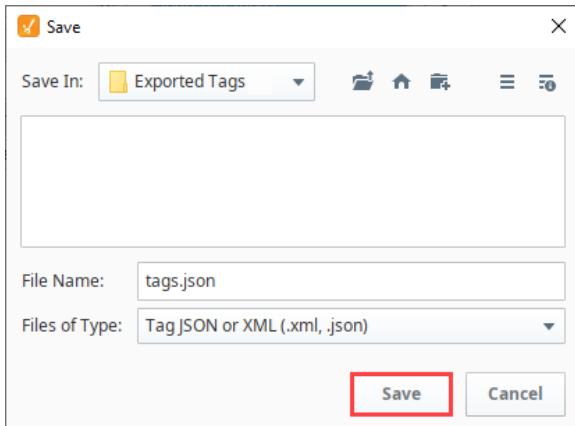
If you select a UDT instance to export, the UDT definition is not automatically included. You must export the definition as a separate file by clicking on the UDT Definitions tab.

When later importing these UDTs, it is recommended to import UDT definitions before importing any instances.

2. On the **Tag Browser** toolbar, click the More Options menu  to open the dropdown. Select **Export Tags**.



3. The **Save** window will open. Specify the folder where you want to save your exported Tag files, and then click **Save**. Ignition will export Tags by default to a **.json** file.



Import Tags

You can import Tags to a individual folder or under the Tags tab. To import Tags under the Tags tab, you can click the Tags tab or the empty space at the bottom of the Tag Browser below all your Tags. When importing Tags, you need to tell Ignition how to handle duplicate Tags. If any of the Tags being imported already exist in the folder you specify, Ignition can abort the import, overwrite the Tags, rename them, ignore them, or merge them. This is called a collision policy.

On this page ...

- [Export Tags](#)
- [Import Tags](#)
- [Tag File Formats](#)
 - [CSV Format](#)
 - [Property Values in the CSV Import](#)
 - [Tag Properties](#)
- [JSON Example](#)
- [XML Example](#)



INDUCTIVE
UNIVERSITY

Importing and Exporting Tags

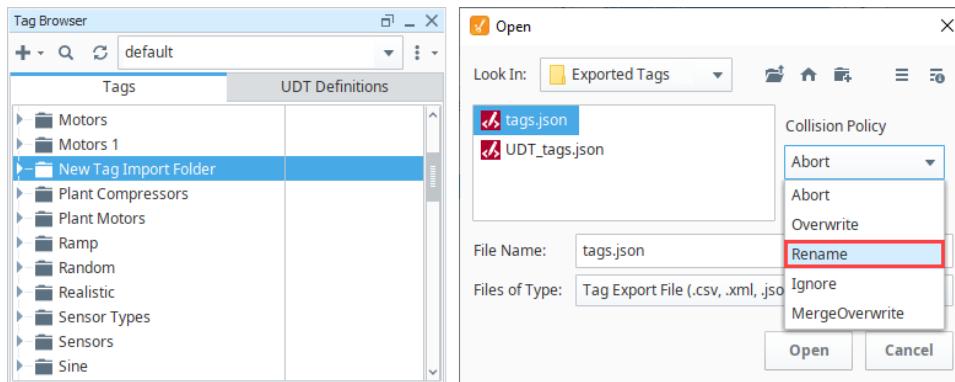
[Watch the Video](#)

Collision Policy Options Table

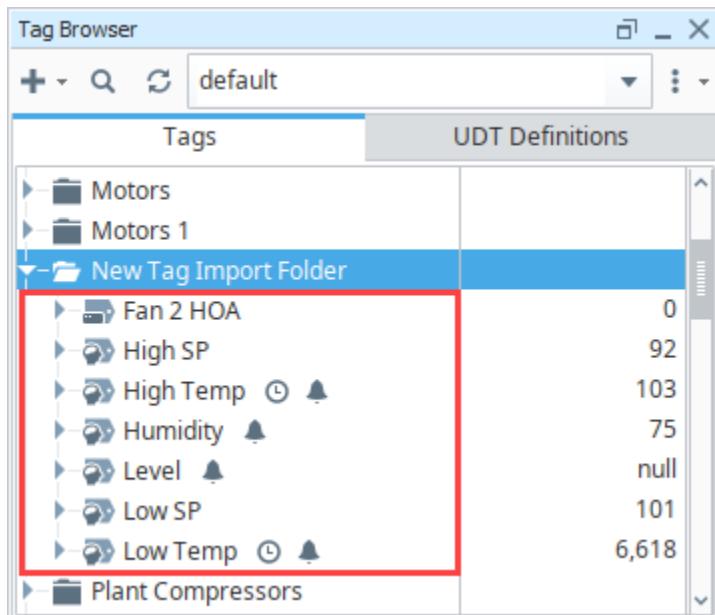
Policy	Description
Abort	Aborts the import if duplicate Tags are found.
Overwrite	Overwrites any Tags in the folder that have the same name as Tags being imported. Note this is a complete overwrite of the Tag.
Rename	Renames any duplicate Tags.
Ignore	Ignores duplicate Tags and imports only those that are unique.
MergeOverwrite	Overwrites the Tag with the exception of any properties that aren't defined in the import folder. Those properties will be merged.

To import Tags, do the steps that follow.

1. In the Tag Browser toolbar, right click on your folder and select **Import Tags**.
2. Specify the folder you want to import your Tags from, and choose a previously exported file either **.json**, **.xml**, or **.csv** file type.
3. Choose a Collision Policy, which indicates how Ignition will deal with duplicate Tags.
4. Click **Open** to import the Tags.



5. The Tags now appear in the Tag Browser in the folder you indicated for the import.



Tag File Formats

Tags can be imported from CSV, JSON, and XML. Tags can only be exported in XML or JSON. There are many configuration settings for Tags than what is displayed in a JSON or XML export file. The Tag export feature only exports the configuration properties that have been edited in at least one of the Tags in the selected export folder. Therefore, to ensure the desired configuration setting is available in the export file, at least one Tag within the selected export folder must have that configuration property changed

CSV Format

Importing

Ignition supports importing tags from a CSV format. Details of the format are below, if you expand "CSV Example Format". This format can contain tag types, OPC paths, and most tag properties. One difference between the CSV format and the XML and JSON format is that the CSV format does not include support for alarm configurations. Alarms can certainly be added to tags in the Ignition Designer after tags have been imported from CSV, but alarms cannot be defined directly in the CSV.

Below is an example of the legacy CSV format, which contains a couple of different tag types, purely for demonstrative purposes. It contains:

- An OPC Tag
- A Folder
- An OPC Tag located in a folder
- A Derived Tag
- An Expression Tag
- A Memory Tag
- A Query Tag

```
Path,Name,Owner,TagType,DataType,Value,Enabled,AccessRights,OPCServer,OPCItemPath,ScanClass,DriverName,
ScaleMode,RawLow,RawHigh,ScaledLow,ScaledHigh,ClampMode,ScaleFactor,Deadband,DeadbandMode,FormatString,
EngUnit,EngLow,EngHigh,EngLimitMode,Tooltip,Documentation,ExpressionType,Expression,OPCWriteBackServer,
OPCWriteBackItemPath,SQLBindingDatasource,HistoryEnabled,PrimaryHistoryProvider,HistoricalScanclass,
HistoricalDeadband,HistoricalDeadbandMode,InterpolationMode,HistoryMaxAgeMode,HistoryMaxAge,
HistoryTimestampSource,UDTParentType,PersistValue,SourceDataType,SourceTagPath,SQLBindingPollRate,
Permissions
# version=1
,_types_,,6,,TRUE,Read_Write,,,Default,,0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,0,100,0,,,0,,,,FALSE,,,
Default Historical,0.01,0,3,0,1,0,,FALSE,,,
,A Folder,,6,2,,TRUE,Read_Write,,,Default,,0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,0,100,0,,,0,,,,,
FALSE,,Default Historical,0.01,0,3,0,1,0,,FALSE,,,
A Folder/,OPC in a folder,,0,2,,TRUE,Read_Write,,,Default,,0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,
0,100,0,,,0,,,,FALSE,,Default Historical,0.01,0,3,0,1,0,,FALSE,,,
,Derived Tag,,13,2,100,TRUE,Read_Write,,,Default,,0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,0,100,0,,,,
0,,,,FALSE,,Default Historical,0.01,0,3,0,1,0,,FALSE,,,[~]Expression Tag,,,
,Expression Tag,,1,2,100,TRUE,Read_Write,,,Default,,0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,0,100,0,,,1,"//This is an expression
100".,,,FALSE,,Default Historical,0.01,0,3,0,1,0,,FALSE,,,
,Memory Tag,,1,7,I'm a memory Tag,TRUE,Read_Write,,,Default,,0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,
0,100,0,,,0,,,,FALSE,,Default Historical,0.01,0,3,0,1,0,,FALSE,,,
,OPC Tag,,0,2,,TRUE,Read_Write,Ignition OPC-UA Server,[devicename]folder/path,Default,,,
0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,0,100,0,,,0,,,,FALSE,,Default Historical,0.01,0,3,0,1,0,,,
FALSE,,,
,Query Tag,,1,2,,TRUE,Read_Write,,,Default,,0,0,100,0,10,0,0,1.00E-04,0,"#,##0.##",,0,100,0,,,2,"/*Some
Query*/
SELECT 100".,,,FALSE,,Default Historical,0.01,0,3,0,1,0,,FALSE,,
```

Property Values in the CSV Import

The following table shows the configuration property names and values contained in legacy CSV Tag import files. Tags were overhauled in Ignition 8.0, so the properties listed here are not the same as Tags export from an Ignition 8.0+ system. For modern tag properties, see the [Tag Properties](#) page.

Tag Properties			
Property Name	Type	Values (if applicable)	Description
Value			The value of the Tag, dependent on the data type.
Data Type	Int	0 - Int1 1 - Int2 2 - Int4 3 - Int8	

		4 - Float4 5 - Float8 6 - Boolean 7 - String 8 - DateTime 9 - DataSet	
Enabled	Boolean	true/false	
Tagtype	Int	0 - OPC Tag 1 - DB Tag (see ExpressionType) 2 - Client Tag 6 - Folder 13 - Derived Tag	Determines the type of the tag. A value of one is a "DB Tag", which is either a Memory Tag, Query Tag, or Expression Tag, depending on the value of the ExpressionType field.
ExpressionType	Int	0 - None 1 - Expression 2 - SQL Query	Used in conjunction when the TagType is set to 1 (DB Tag), otherwise this field is ignored.
AccessRights	Int	0 - Read Only 1 - Read/Write 2 - Custom	If custom, will be defined by a Permissions Tag.
OPCServer	String		
OPCItemPath	String		
OPCWriteBack Server	String		Write back target for expression Tags.
OPCWriteBack ItemPath	String		
ScaleMode	Int	0 - Off 1 - Linear 2 - Square Root 3 - Exponential Filter	
ScaleFactor	Float		For exponential filter.
RawLow	Float		Defines scale range.
RawHigh	Float		
ScaledLow	Float		
ScaledHigh	Float		
ClampMode	Int	0 - None 1 - Low 2 - High 3 - Both	
Deadband	Float		
DeadbandMode	Int	0 - Absolute 1 - Percentage	
FormatString	String		
EngUnit	String		
EngLow	Float		
EngHigh	Float		
EngLimitMode	Int	0 - None 1 - Low 2 - High 3 - Both	
Tooltip	String		
Documentation	String		
DriverName	String		Used for external Tags.
ScanClass	String		The export will only include the name of the Scanclass, not the configuration of the Scanclass itself. A Scanclass with the same name needs to already exist on the Gateway that the Tags are being imported to, prior to importing them.
HistoryEnabled	Boolean	true/false	
PrimaryHistory Provider	String		The history provider to use if storing history.
HistoricalDeadband	Float		
HistoricalDeadb	Int		

andMode		0 - Absolute 1 - Percentage	
HistoricalScanClass	String		
InterpolationMode	Int	0 - Discrete 2 - Analog (deadband) 3 - Analog (compressed)	How values are interpolated. 2 exists for backwards compatibility (and is equivalent to 1), but only 0 or 3 should be used in the future.
HistoryTimeStampSource	Int	0 - System 1 - Value	
HistoryMaxAgeMode	Int	0 - Unlimited 1 - Limited	
HistoryMaxAge	Int		Max cycles between storage.
UDTParentType	String		The path to the parent UDT type. Used by sub-types and instances.

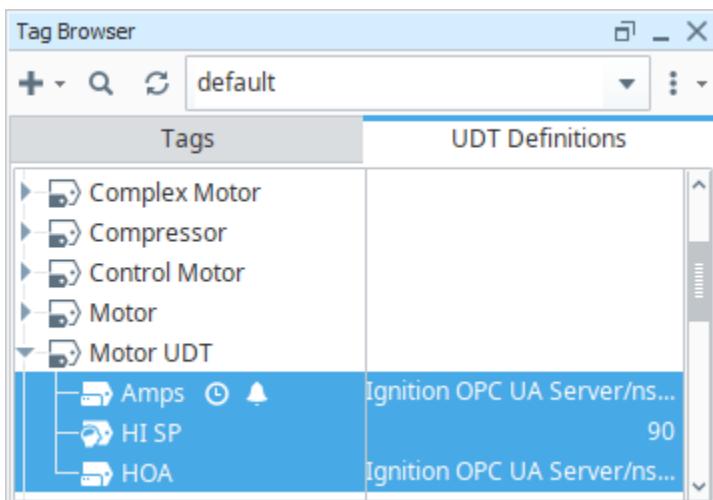
Exporting

Although Ignition can import tags from a CSV format, Ignition does not export tags to a CSV format. Since multiple alarms can be configured per tag, the XML or JSON formats provide a much better format to allow for the tree structures needed to fully represent Ignition tags.

JSON Example

In this example, we exported three Tags from the Motor UDT in JSON format.

- Amps - Expression Tag with an Alarm and History enabled
- HI SP - Memory Tag which is bound to a parameter to an OPC Tag
- HOA - OPC Tag



The following exported Tag file is in JSON format. As you browse through the JSON file, you will see the Tag properties and configuration settings for each of the three Tags listed above.

Below is an example of a JSON format tag export. Descriptions on the various properties can be found on the [Tag Properties](#) page.

Exported Tags in JSON Format
<pre>{ "tags": [{ "opcItemPath": { "bindType": "parameter", "binding": "ns\u002f003d1;s\u002f003d[Dairy]_Meta:Overview/Motor {MotorNumber}/Amps" }, "valueSource": "opc", "historyProvider": "MySQL", "alarms": [{ "alarmType": "expression", "expression": "value > 90" }] }] }</pre>

```

    {
        "mode": "BelowValue",
        "setpointA": 25.0,
        "name": "Low Amps",
        "priority": "Critical",
        "displayPath": {
            "bindType": "Expression",
            "value": "Motor{MotorNumber}"
        }
    },
    {
        "mode": "AboveValue",
        "name": "High Amps",
        "priority": "Critical",
        "setpointA": {
            "bindType": "Expression",
            "value": "{[.]HI SP}"
        }
    }
],
"name": "Amps",
"historyEnabled": true,
"tagType": "AtomicTag",
"opcServer": "Ignition OPC UA Server"
},
{
    "valueSource": "memory",
    "name": "HI SP",
    "value": 90,
    "tagType": "AtomicTag"
},
{
    "opcItemPath": {
        "bindType": "parameter",
        "binding": "ns\u003d1;s\u003d[Dairy]_Meta:Overview/Motor {MotorNumber} /HOA"
    },
    "valueSource": "opc",
    "name": "HOA",
    "tagType": "AtomicTag",
    "opcServer": "Ignition OPC UA Server"
}
]
}

```

XML Example

In this example, we exported the same three Tags from our Tag Browser, that were also used in the JSON example, in XML format. As you browse through the XML file, you will see the Tag properties and configuration settings for each of the same three Tags.

Below is an example of an XML format tag export. Descriptions on the various properties can be found on the [Tag Properties](#) page.

Exported Tags in XML Format
<pre> <Tags MinVersion="8.0.0" locale="en_US"> <Tag name="Amps" type="AtomicTag"> <Property name="opcItemPath" boundValueType="parameter">ns=1;s=[Dairy]_Meta:Overview/Motor {MotorNumber}/Amps</Property> <Property name="valueSource">opc</Property> <Property name="historyProvider" datatype="String">MySQL</Property> <CompoundProperty name="alarms"> <PropertySet> <Property name="mode">3</Property> <Property name="setpointA">25</Property> <Property name="name">Low Amps</Property> <Property name="priority">4</Property> </PropertySet> </CompoundProperty> </Tag> </Tags> </pre>

```

<Property name="displayPath" bindtype="Expression">Motor{MotorNumber}</Property>
</PropertySet>
<PropertySet>
  <Property name="mode">2</Property>
  <Property name="name">High Amps</Property>
  <Property name="priority">4</Property>
  <Property name="setpointA" bindtype="Expression">{[.]HI SP}</Property>
</PropertySet>
</CompoundProperty>
<Property name="historyEnabled" datatype="Boolean">true</Property>
<Property name="opcServer">Ignition OPC UA Server</Property>
</Tag>
<Tag name="HI SP" type="AtomicTag">
  <Property name="valueSource">memory</Property>
  <Property name="value">90</Property>
</Tag>
<Tag name="HOA" type="AtomicTag">
  <Property name="opcItemPath" boundValueType="parameter">ns=1;s=[Dairy]_Meta:Overview/Motor
{MotorNumber}/HOA</Property>
  <Property name="valueSource">opc</Property>
  <Property name="opcServer">Ignition OPC UA Server</Property>
</Tag>
</Tags>

```

Related Topics ...

- [Tag Data Types](#)
- [Tag Properties](#)

Alarming

Overview

Alarming is a core feature of the Ignition platform. It provides the functionality and flexibility to configure your alarms, provides up-to-date status of alarms, stores alarm history, builds the logic for how, why, and when alarm notifications are delivered, manages alarm notifications for user groups, and sends Email, SMS or Voice notifications. With all these features and functions in Alarming, you can easily create alarms, and design and manage your alarm notifications any way you choose.

- **Alarm Settings** - There are a host of alarm settings so you can build and change the behavior of your alarm.
- **Alarm Status** - Displays current status of alarms.
- **Alarm Journal** - Stores historical information about alarms.
- **Alarm Notification** - Lets you select the delivery channel for how alarms notifications are sent via [Email](#), [SMS](#), or [Voice](#).
- **Users, Schedules, and Rosters** - Defines the users who will receive alarms based on [user schedules](#) and [on-call rosters](#).
- **Alarm Notification Pipelines** - Lets you define how alarm notifications are sent out and acknowledged.

Alarms can be configured on Tags or OPC items in [SQL Bridge \(Transaction Groups\)](#). The different [Tag types](#) that you can configure alarms on include Memory Tags, Query Tags, Expression Tags, as well as Tags inside of a UDT. You can also put alarms on [System Tags](#) that Ignition inherently provides such as the Gateway Performance, CPU Usage, and more.

Configuring Alarms

Alarm configuration in Ignition is flexible and highly customizable to your needs. You can configure alarms with one alarm on a Tag or multiple alarms on a Tag. You can add alarms in UDTs so every instance of that Tag will automatically have alarms configured when a new instance of your Tag is created. You can use the alarm settings to create alarms that equal or don't equal a setpoint, above or below a setpoint, between setpoints, outside setpoints, dynamic setpoints, out of range, bad quality, etc. Alarms can be configured for any alarm condition imaginable.

More information about configuring alarms can be found in [Alarming Properties](#) and [Configuring Alarms](#) pages.

The image below shows an alarm configured on an OPC Tag. You can see that an alarm has quite a few properties including alarm mode settings where you can set specific alarm attribute values.

On this page ...

- [Overview](#)
- [Configuring Alarms](#)
- [How to Monitor Alarm Status](#)
 - [Alarm Status Tags](#)
- [How to View Alarm History](#)
- [Alarm Notifications](#)
 - [On-Call Rosters](#)
 - [Alarm Notification Pipelines](#)

Ramp8 > Alarms

OPC Alarm - Outside Setpoints, Critical < OPC Alarm

		A ↓	2 ↓							
Main										
Name	OPC Alarm									
Enabled	true									
Priority	Critical									
Timestamp Source	System									
Label										
Display Path										
Ack Mode	Manual									
Notes										
Ack Notes Required	false									
Shelving Allowed	true									
Alarm Mode Settings										
Mode	Outside Setpoints									
Low Setpoint	150									
Low Inclusive	true									
High Setpoint	300									
High Inclusive	true									
Any Change	false									
Deadbands and Time Delays										
Deadband	0									
Deadband Mode	Absolute									
Active delay (seconds)	0									
Clear delay (seconds)	0									
Notification										
Ack Pipeline										
Active Pipeline										
Clear Pipeline										
Email Notification Properties										
Custom Message										
Custom Subject										
SMS Notification Properties										
Custom Message										

How to Monitor Alarm Status

Setting up the Alarm Status Table is quick and easy. Drag the [Alarm Status Table](#) component onto your window, and the current alarms are displayed immediately into one view. The Alarm Table is highly customizable and alarms can be configured to show active, unacknowledged, cleared, and acknowledged alarms. You can [Acknowledge](#) or [Shelve](#), and you can [filter on Alarm Status properties](#), show or hide alarm property information, reorganize columns, or view alarm details, notes, and history of an alarm.

Active Time	Display Path	Priority	Current State
<input checked="" type="checkbox"/> 5/30/19, 11:23 AM	Motors/Motor 4/Amps/Low Amps	Critical	Active, Unacknowledged
<input type="checkbox"/> 5/30/19, 11:25 AM	Motors/Motor 2/Amps/Low Amps	Critical	Active, Unacknowledged
<input type="checkbox"/> 5/30/19, 11:26 AM	Ramp/Ramp8/OPC Alarm	Critical	Active, Unacknowledged
<input type="checkbox"/> 5/30/19, 10:33 AM	Turbine Number 200 located at Livermore, CA	High	Active, Unacknowledged
<input type="checkbox"/> 5/30/19, 10:33 AM	Turbine Number 300 located at Fresno	High	Active, Unacknowledged
<input type="checkbox"/> 5/30/19, 11:13 AM	Motors/Motor 2/Amps/Low Amps	Critical	Cleared, Unacknowledged
<input type="checkbox"/> 5/30/19, 11:14 AM	Motors/Motor 2/Amps/Low Amps	Critical	Cleared, Unacknowledged

Details Notes

Config Properties

- + **On Active**
- mode Below Setpoint
- setpointA 30
- Event Value 28
- name Low Amps
- Event Time 5/30/19, 11:23 AM
- priority Critical

Acknowledge Shelf

Alarm Status Tags

Ignition provides a set of System Tags to view information about the Ignition server which includes four Tags that count the number of alarms in each state. A quick way to see if any alarms are currently active and get an alarm count is to add a Label component on the Navigation window. The four system alarm states are:

- Active and Unacknowledged
- Active and Acknowledged
- Clear and Acknowledged
- Clear and Unacknowledged

In the **Tag Browser** of the Designer, scroll down to the **System > Gateway > Alarming** folder. You can see all four of the system Tags that Ignition provides. You can also see how many alarms are currently **Active and Acked**, **Active and Unacked**, **Clear and Acked**, and **Clear and Unacked**.

These system Tags can easily be used to [visualize all alarms in the system](#).

Tag	Value	Data Type	Traits
Active and Acked	3	Integer	
Active and Unacked	4	Integer	
Clear and Acked	6	Integer	
Clear and Unacked	40	Integer	

How to View Alarm History

The [Alarm Journal](#) stores historical information about alarms in a database. It stores basic data about alarms that have occurred, such as their source and timestamp, associated data on an alarm, and the values of an alarm's properties at the time the event occurred. It captures all status changes for each alarm, as well as acknowledgement to an external SQL database of your choosing. To begin viewing alarm history, all you need to do is create an [Alarm Journal Profile](#) in the Gateway webpage.

Like the Alarm Status Table, the Alarm Journal enables you to [filter on alarm history properties](#). The alarms are color coded so you know what each status represents. The [Date Range](#) is a very common filter type since users typically want to filter for alarm events within a specific period of time.

Event Time	Display Path	Event State	Priority	Current State	Label
5/30/19, 11:45 AM	Motors/Motor 3/Amps/Low Amps	Active	Critical	Active, Unacknowledged	Low Amps
5/30/19, 11:45 AM	Motors/Motor 3/Amps/Low Amps	Ack	Critical	Cleared, Acknowledged	Low Amps
5/30/19, 11:45 AM	Ramp/Ramp8/OPC Alarm	Active	Critical	Active, Unacknowledged	OPC Alarm
5/30/19, 11:45 AM	Ramp/Ramp8/OPC Alarm	Ack	Critical	Cleared, Acknowledged	OPC Alarm
5/30/19, 11:45 AM	Motors/Motor 3/Amps/Low Amps	Active	Critical	Active, Unacknowledged	Low Amps
5/30/19, 11:45 AM	Motors/Motor 3/Amps/Low Amps	Ack	Critical	Cleared, Acknowledged	Low Amps
5/30/19, 11:45 AM	Ramp/Ramp8/OPC Alarm	Active	Critical	Active, Unacknowledged	OPC Alarm
5/30/19, 11:45 AM	Ramp/Ramp8/OPC Alarm	Ack	Critical	Cleared, Acknowledged	OPC Alarm
5/30/19, 11:45 AM	Motors/Motor 3/Amps/Low Amps	Active	Critical	Active, Unacknowledged	Low Amps
5/30/19, 11:45 AM	Motors/Motor 3/Amps/Low Amps	Ack	Critical	Cleared, Acknowledged	Low Amps
5/30/19, 11:44 AM	Motors/Motor 3/Amps/Low Amps	Active	Critical	Active, Unacknowledged	Low Amps
5/30/19, 11:44 AM	Motors/Motor 3/Amps/Low Amps	Ack	Critical	Cleared, Acknowledged	Low Amps
5/30/19, 11:44 AM	Ramp/Ramp8/OPC Alarm	Active	Critical	Active, Unacknowledged	OPC Alarm
5/30/19, 11:44 AM	Ramp/Ramp8/OPC Alarm	Ack	Critical	Cleared, Acknowledged	OPC Alarm
5/30/19, 11:44 AM	Motors/Motor 1/Amps/Low Amps	Active	Critical	Active, Unacknowledged	Low Amps
5/30/19, 11:44 AM	Motors/Motor 1/Amps/Low Amps	Ack	Critical	Cleared, Acknowledged	Low Amps
5/30/19, 11:44 AM	Ramp/Ramp8/OPC Alarm	Active	Critical	Active, Unacknowledged	OPC Alarm

7,287 events

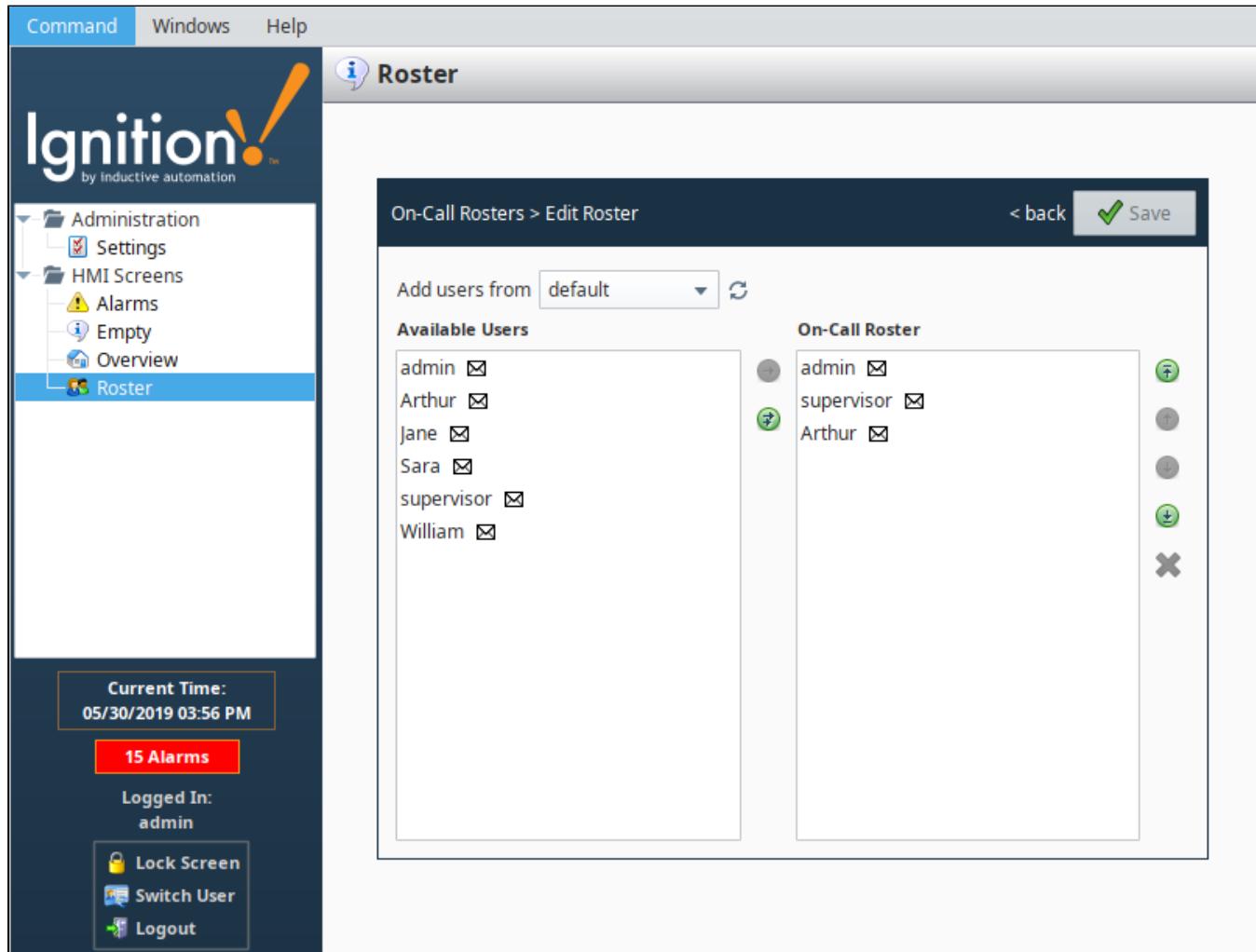
Alarm Notifications

Alarms can also generate notifications that are delivered to users allowing Ignition to immediately communicate events and problems to your users. Alarm Notification Pipelines control how and when notifications are sent to users. You can select the delivery channel for how alarms are sent: [Email](#), [SMS](#), or [Voice](#). The notification system has access to Ignition's Authentication Profiles so users can easily be added to notification [On-Call Rosters](#). [Schedules](#) can be created allowing users to receive notifications only when on-schedule, so there is no need to worry about notifying a supervisor or manager when they are not on-site, or if it's in the middle of the night. Pipelines coupled with on-call rosters and schedules allow you to build your own custom alarm notification process.

On-Call Rosters

The [On-Call Roster](#) is a collection of users that are notified when an alarm occurs. When an alarm is triggered, a notification is sent to a designated On-Call Roster where it evaluates the users schedules, and only notifies those users that have an active schedule.

Roster Management from the Vision Client Window

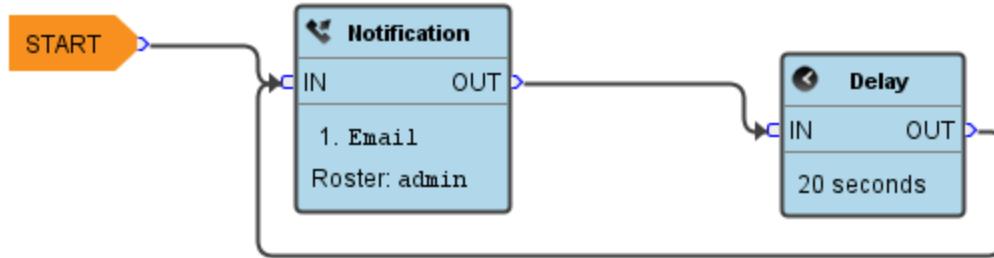


Alarm Notification Pipelines

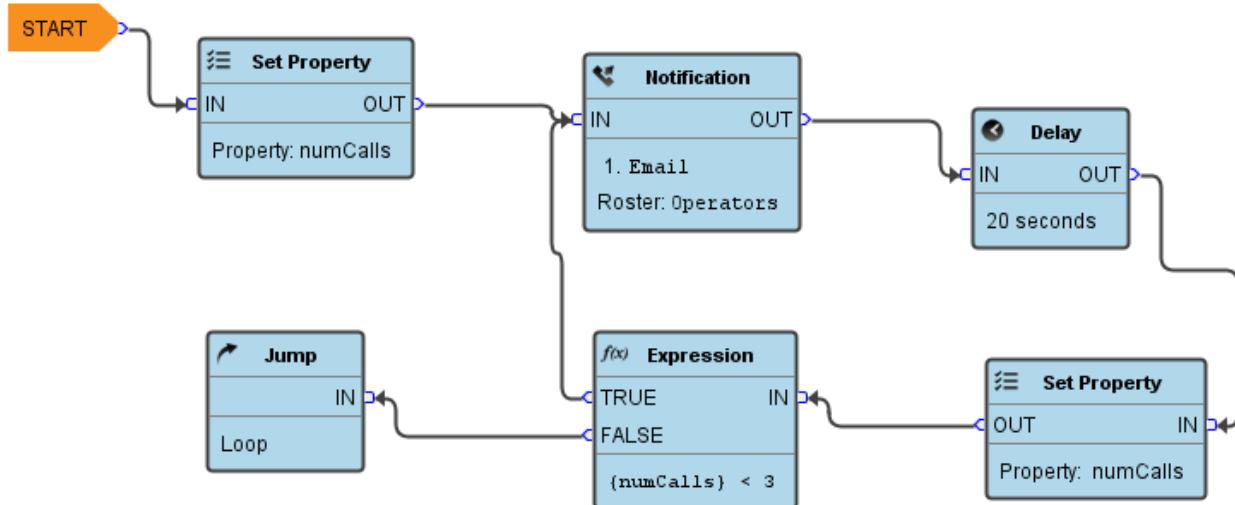
The [Alarm Notification Pipeline](#) feature is an innovative tool that lets you easily create routes for your alarms. By designing your own alarm notification routing, you have control of what happens when an alarm goes active, when an alarm is sent out, and who receives the alarm notification.

The alarm notification pipeline has a simple drag-and-drop interface so you can build various types of alarm logic. Its built-in [pipeline block](#) functionality lets you delay sending alarm notifications, [escalate](#) higher priority alarms to a different group of users, [consolidate](#) multiple alarms so recipients aren't flooded with messages, and distribute specific alarm types to different contact groups.

Alarm notification pipelines can be very simple to very complex. In this simple notification pipeline, when an alarm is triggered, the people listed in the [On-call Roster](#) are notified via Email. If no one acknowledges the alarm in 20 seconds, the alarm notification is routed back to the same users listed in the On-Call Roster.



In this more complex alarm notification pipeline, if an operator doesn't respond to the alarm after three attempts, the pipeline jumps it to another pipeline (possibly an escalation pipeline).



To learn more about building your own pipelines, go to [Alarm Notification Pipelines](#).

Related Topics ...

- Configuring Alarms
- Using Alarm Status Tags in Vision
- Alarm Pipeline Designer Interface
- Pipeline Blocks

In This Section ...

Alarm Journal

By default, current alarm data is only stored in memory, and a finite number of events are retained for each alarm. Fortunately, Ignition can easily be configured to store alarm data into a SQL database with an Alarm Journal Profile. The journal can store basic data about alarms that have occurred, such as their source and timestamp, associated data on the alarm, and the values of the alarm's properties at the time the event occurred. The Alarm Journal is used by the **Alarm Journal Table** component, and can be accessed through scripting functions and direct database queries.

The Gateway can have more than one Alarm Journal. Alarm Journals have options to filter which Alarms are stored in the journal, therefore, by having more than one alarm journal configured on the Gateway, it is possible to store some alarms in one journal, and different alarms in another journal. Once configured, the journal can be accessed by the **Alarm Journal Table** component, scripting functions, or direct database queries.

Alarm Journals can store data in one of three ways, and store data indefinitely unless a Data Pruning value is set:

- In a database, using an existing database connection from the Gateway
- Remotely, using another Ignition Gateway's Alarm Journal profile
- Internally, storing alarm information into the Ignition install directory

It is strongly encouraged to set a Data Pruning value for Internal Alarm Journal Profiles. Otherwise, it could cause your computer to run out of hard drive space.

Note: You must have an [Alarm Journal Profile](#) created and have a valid [database connection](#) to use the Alarm Journal Table.

Creating an Alarm Journal Profile

In Ignition, an Alarm Journal stores historical information about alarms in a database. It can store basic data about alarms that have occurred, such as source and timestamp, along with associated data on the alarm, and the values of the alarm's properties at the time the event occurred. You can choose to store your alarm data on an external database or automatically send it to a remote gateway's Alarm Journal to be logged. Both options are described in the sections below.

Create a single Alarm Journal Profile to store all of your alarms, or create multiple journals to store alarms across multiple databases. Each journal stores alarms based on the filters you set up and can prune data automatically after a set time limit.

Create an Alarm Journal to Log Events to an External Database

1. Go to the **Config** section of the Gateway webpage.
2. Choose **Alarming > Journal** from the menu on the left.
3. Look for the blue arrow and click on [Create new Alarm Journal Profile....](#) The Alarm Journal Profiles screen will be displayed.

Name	Datasource	
Journal	Mysql	delete edit
Packaging_Area	Mysql	delete edit

4. You have the option of logging alarm journal events to an external database, logging locally, or sending them to a remote gateway's Alarm Journal. In this example, select **Database**, and click **Next**. (Configuring alarm journal events to be sent to a remote gateway is addressed in [Create an Alarm Journal to Log Events to a Remote Gateway](#) section on this page.)

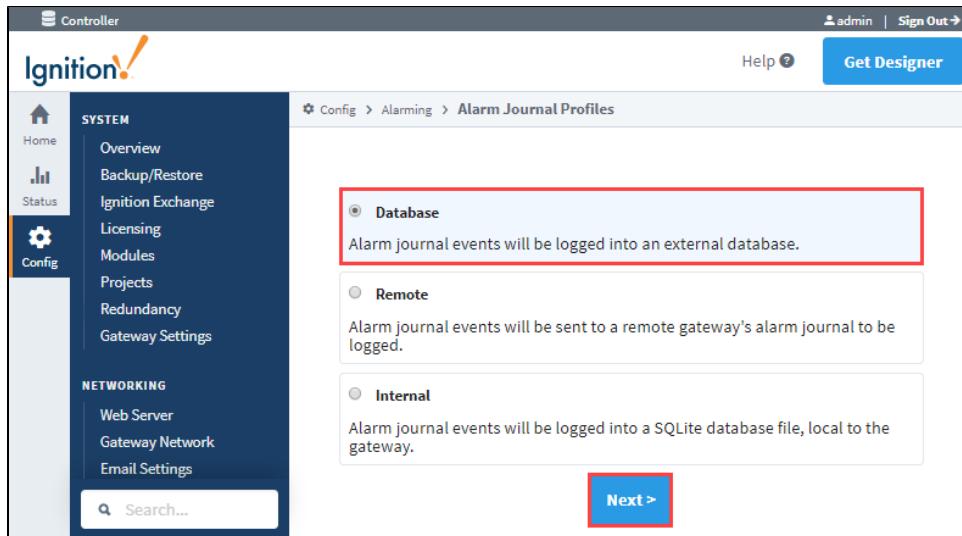
On this page ...

- [Creating an Alarm Journal Profile](#)
 - [Create an Alarm Journal to Log Events to an External Database](#)
 - [Create an Alarm Journal to Log Events to a Remote Gateway](#)
 - [Create an Internal Alarm Journal to Log Events Locally](#)
- [Third Party Accessibility](#)
- [Alarm Journal Component](#)
- [Journal Properties](#)
 - [Table Definitions](#)



Create Alarm Journal Profile

[Watch the Video](#)



5. Enter the **Name** of your alarm journal profile. The default name is 'Journal.' Most of the fields have default settings. Refer to the [journal properties table](#) below for setting descriptions, and update as necessary. Click the **Create New Alarm Journal Profile** button at the bottom of the page. Once completed, the tables will be created for you once an alarm event occurs.

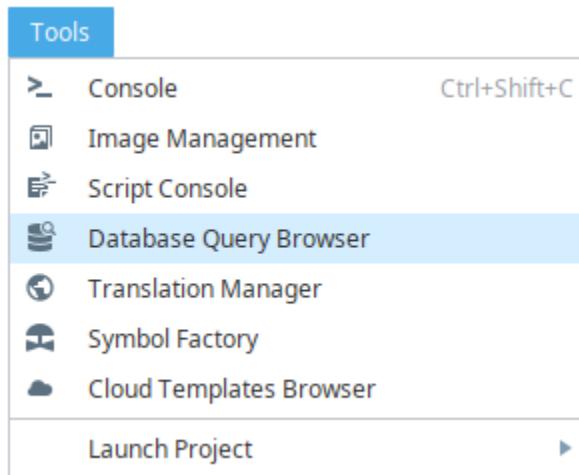
Main	
Name	Packaging_Area
Description	
Enabled	<input checked="" type="checkbox"/> (default: true)

Main	
Datasource	MySQL
Events will be stored to this datasource.	
Minimum Priority	Low
Only events equal to or greater than the specified priority will be stored. (default: Low)	
Store Shelved Events	<input type="checkbox"/> If enabled, events generated by "shelved" alarms will still be written to the journal system. (default: false)
Use Store and Forward	<input checked="" type="checkbox"/> If enabled, journaled events will be stored through the store and forward system. If not enabled, they will be stored directly against the database. (default: true)

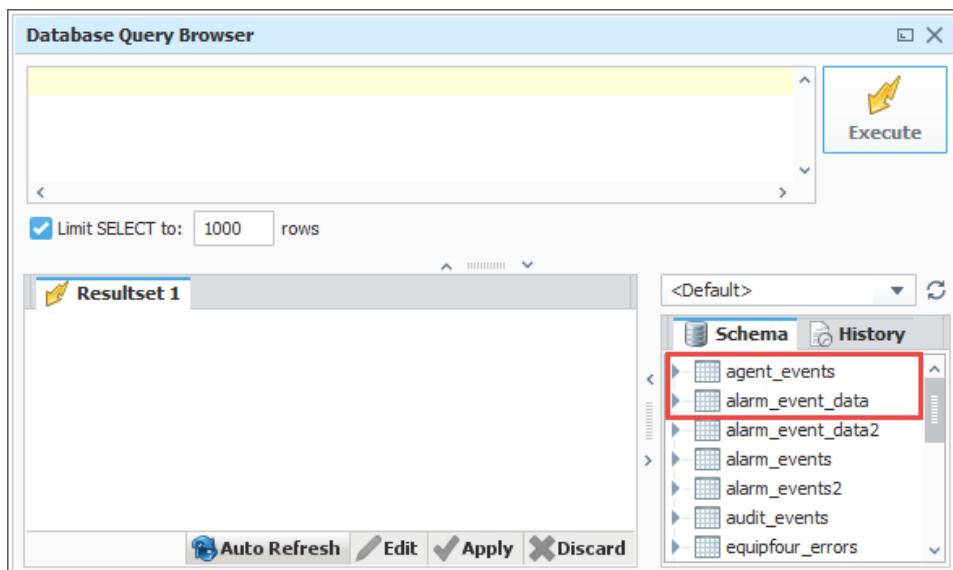
Note: If you only have one alarm journal specified on your Gateway, then you do not need to specify the journal name on the Journal Name property. Ignition will set this for you. If you have more than one alarm journal created, then you need to provide the name of the journal you'd like to query in the Journal Name component property of the Property Editor.

6. It's a good idea to trigger a test alarm and verify that Ignition automatically created the tables in the database after your Alarm Journal Profile is created.

You can easily verify the Alarm Journal tables from the Ignition **Designer**. Open the journal table and go to the menu bar and select **Tools > Database Query Browser**.



The two tables that Ignition automatically creates are the `alarm_events` and `alarm_event_data`. Alarm events consist of two main types of data: the primary information about the alarm, such as transition state, time, etc., and the event data. If you don't see these two tables in your database, make an alarm occur after the Alarm Journal Profile is created. Make the alarm Active, take it back to Clear, Acknowledge it, and you will start seeing information come into these tables.



Double click to expand the `alarm_events` to see all the events that occurred when the alarms are Active, Cleared, and Acknowledged.

The screenshot shows the Database Query Browser interface. In the top-left, a query window contains the SQL command: `SELECT * FROM alarm_events`. Below it, a message says "1000 rows fetched in 0.019s". To the right is a "Resultset 1" table with columns: id, eventid, source, and displaypath. The table lists 12 rows of alarm event data. At the bottom are buttons for Auto Refresh, Edit, Apply, and Discard. On the far right is a "Schema" browser showing the database structure, with the `alarm_event_data` table highlighted.

The `alarm_event_data` is all the associated data that is associated with each alarm.

This screenshot is similar to the previous one but shows the results of a query on the `alarm_event_data` table. The table has columns: id, propname, dtype, intvalue, floatvalue, and strvalue. The schema browser on the right highlights the `alarm_event_data` table and its columns: dtype, floatvalue, id, intvalue, propname, and strvalue.

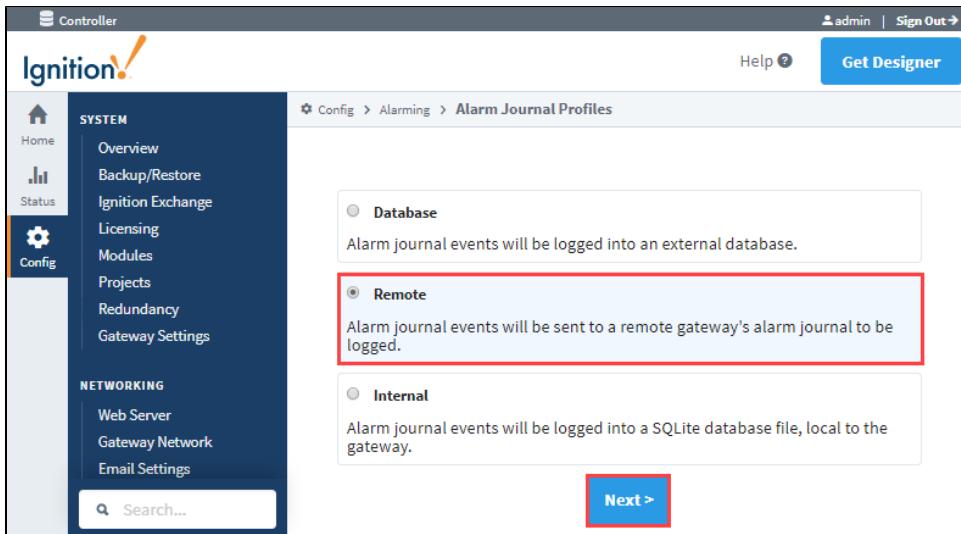
You will want to look into the [Database Query Browser](#) to simply verify that the information is there. Once the information is there, close the Database Query Browser.

Note: If the tables were not created, check the Gateway Console page for any errors.

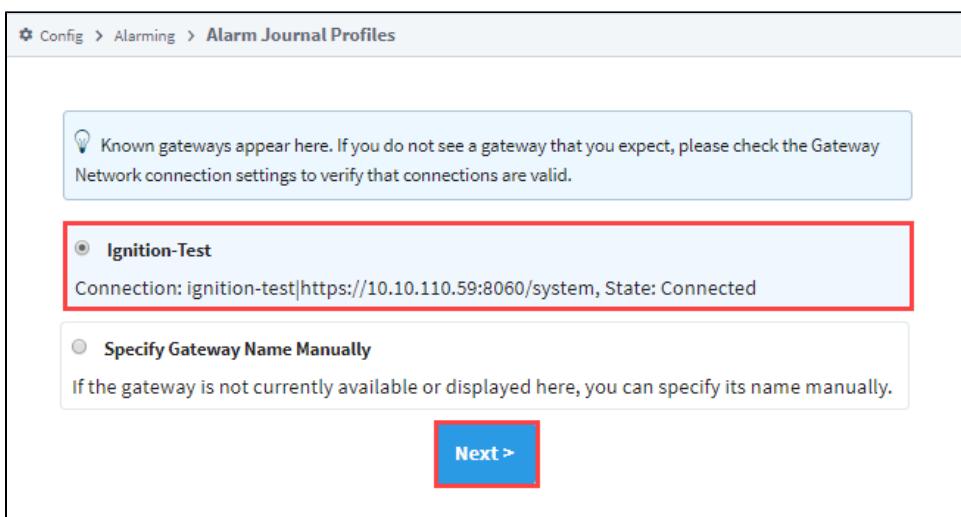
Create an Alarm Journal to Log Events to a Remote Gateway

Just like configuring alarm journal events to be logged into a database, it is done from the Gateway Webpage, **Config > Alarming > Journal**.

1. To have your alarm journal events automatically sent to a remote gateway's alarm journal, select **Remote**, and click **Next**.



2. A list of known Gateways will be displayed. If you don't see a gateway that you expected to see, check your Gateway Network settings to verify that the connections are valid. You also have the option to specify a gateway manually. This example selects a valid gateway. Click **Next**.



3. If an Alarm Journal exists on the remote gateway, the fields will autopopulate. The name of the gateway and the Alarm Journal Profile name will appear in the **Name** field prefaced with the alarm journal profile name,(i.e., Ignition_Test_Journal), as shown in the following example. Click **Create New Alarm Journal Profile**.

Config > Alarming > Alarm Journal Profiles

Main	
Name	Ignition_Test_Journal <input type="button" value="edit"/>
Description	<input type="text"/>
Enabled	<input checked="" type="checkbox"/> (default: true)
Remote Gateway	
Gateway Name	Ignition-Test
Alarm Journal	Journal
Create New Alarm Journal Profile	

4. You will receive a successful message stating your new Alarm Journal Profile was created.

Config > Alarming > Alarm Journal Profiles

✓ Successfully created new Alarm Journal Profile "Ignition_Test_Journal"

Name	Type	Description
Ignition_Test_Journal	Remote	<input type="button" value="delete"/> <input type="button" value="edit"/>
Journal	Database	<input type="button" value="delete"/> <input type="button" value="edit"/>
Packaging_Area	Database	<input type="button" value="delete"/> <input type="button" value="edit"/>
→ Create new Alarm Journal Profile...		

Remote Gateway Alarm Journal Properties Table

Main	
Name	The default name, is the name of the Remote Gateway and Alarm Journal.
Description	Description of the journal profile. Optional
Enabled	By default. The journal profile is enabled.
Remote Gateway	
Gateway Name	Name of the Remote Gateway.
Alarm Journal	Name of the Alarm Journal on the Remote Gateway.

Create an Internal Alarm Journal to Log Events Locally

Ignition Gateways can now create an Internal Alarm Journal Profile that stores Journal entries locally.

Go to the Gateway webpage, **Config > Alarming > Journal** to create the Internal alarm journal profile.

1. Click on the the **Create new Alarm Journal Profile...** link.
2. Select **Internal** to have your alarm events logged locally.

The screenshot shows the Ignition configuration interface. On the left, there's a sidebar with 'Config' selected. The main panel shows 'Alarm Journal Profiles' with three options: 'Database', 'Remote', and 'Internal'. The 'Internal' option is highlighted with a red box. At the bottom right of the main panel is a blue 'Next >' button.

3. Enter the name of your alarm journal profile and update any settings as required, then click **Next**.

The screenshot shows the 'Main' configuration screen for an alarm journal profile. It includes fields for Name (set to 'Ignition_Internal_Journal'), Minimum Priority (set to 'Low'), Description (empty), Store Shelved Events (unchecked), and Enabled (checked).

4. You will receive a successful message stating your new Alarm Journal Profile was created.

Internal Alarm Journal Properties Table

Main	
Name	The default name, is the name of the Remote Gateway and Alarm Journal.
Minimum Priority	Only events equal to or greater than the specified priority will be stored.
Description	Description of the journal profile. Optional
Stored Shelved Events	If enabled, events generated by "shelved" alarms will still be written to the journal system. Default is false.
Enabled	By default. The journal profile is enabled.

Third Party Accessibility

Because the Alarm Journal uses a SQL database to log alarm events, any application that has access to the database can retrieve journal data. Alarm events can be made freely available outside of Ignition, and integrated into other software packages. Additionally, other applications can write to the same tables, so Ignition applications can monitor activity in other systems.

Alarm Journal Component

While a SQL query will return data from the journal, the Alarm Journal Table component will automatically do so without manually writing a query. The component can filter on both [Display Path](#) and [Source Path](#), as well as [Date Range](#). The component can be configured to a single Journal Profile, so multiple instances of the component in the same project may look at different profiles.

More information on this component can be found on the [Alarm Journal Table Component](#) page.

Event Time	Display Path	Event State	Event Value	Priority	Current State
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Active	50	Critical	Active, Unacknowledged
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Ack		Critical	Cleared, Acknowledged
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Clear	54	Critical	Cleared, Unacknowledged
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Active	47	Critical	Active, Unacknowledged
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Ack		Critical	Cleared, Acknowledged
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Clear	51	Critical	Cleared, Unacknowledged
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Active	47	Critical	Active, Unacknowledged
6/10/19, 3:15 PM	Motors/Motor 3/Amps/Low Amps	Ack		Critical	Cleared, Acknowledged
6/10/19, 3:14 PM	Motors/Motor 3/Amps/Low Amps	Clear	51	Critical	Cleared, Unacknowledged

3,749 events   

Journal Properties

Main	
Name	The default name is Journal.
Description	Description of the journal profile.
Enabled	By default, the journal profile is enabled.
Datasource	This is the ONLY required setting which must be a valid database connection. Events are stored to this datasource.
Minimum Priority	Only events equal to or greater than the specified priority will be stored. The default is Low. You can set the priority to be: Diagnostic, Low, Medium, High, and Critical.
Store Shelved Events	Not enabled by default. If enabled, events generated by "shelved" alarms will still be written to the journal system.
Use Store and Forward	Enabled by default, which means the alarm journaled events will be stored through the Store and Forward system . If not enabled, they will be stored directly against the database. This system protects data from being lost due to temporary database connectivity issues.
Stored Event Data	
Alarm events consist of two main types of data: the primary information about the alarm, such as transition state, time, etc., and the event data .	
The following settings specify what type of event data is stored:	
Static Config	By default, it is not selected. If selected, will store the values of static alarm configuration. That is, the alarm properties that are <i>not</i> bound. These do not change during evaluation, only when a user modifies them in the Designer, and so they are not stored by default.
Dynamic Config	If selected, will store the values of dynamically bound alarm configuration properties. The value of these properties can change at any time, and the values at the time of the alarm are captured on the alarm event.

Static Associated Data	If selected, will store the values of non-bound associated data (properties created by the user) properties on alarm that do not change during execution.
Dynamic Associated Data	If selected, will store the values of dynamically bound associated data (properties created by the user) properties.

Data Filters

Note:

The following three properties interact via logical AND, meaning an alarm must meet the criteria for all three for it to be logged in the Journal. Thus, if you supply values for all three Data Filter properties, then an alarm must match the **Filter by Alarm Source**, **Filter by Display Path**, and **Filter by Display Path or Source** properties.

Example: If a journal has values for all three properties, and an alarm only meets the requirements for **Filter by Alarm Source** and **Filter by Display Path or Source**, but not **Filter by Display Path**, then the alarm will not be logged to the Journal.

If you want to filter on both the Display Path and Source Path, you would want to use only one of the two following approaches:

- **Filter by Alarm Source and Filter by Display Path**
- Only use **Filter by Display Path or Source**

It is recommended to avoid using all three properties simultaneously on the same Journal.

Filter by Alarm Source	Only events matching the source will be stored. Multiple sources to match can be comma separated. Leave blank to store events from all sources.
Filter by Display Path	Only events matching the display path will be stored. Multiple display paths to match can be comma separated. Leave blank to store events from all display paths.
Filter by Display Path or Source	Only events matching the display path, if defined, will be stored. Multiple matches can be comma separated. If no display path is defined, only events matching the source will be stored. Leave blank to store all events.

Data Pruning

Enable Data Pruning	If selected, data will be deleted after the specified time period as set by the Prune Age and Units below. Default is false.
Note: Since the data is stored directly in a database, an administrator is free to manually delete data at any time.	
Prune Age	The number of Prune Age Units to store data for. i.e., 1 year, 5 hours, etc. The default is 1.
Prune Age Units	The type of Prune Age Unit. Default is Years. You can choose the unit to be Milliseconds, Seconds, Minutes, Hours, Days, Weeks, Months, or Years.

Advanced

These settings let you specify your own table names. This is especially useful when trying to use multiple alarm profiles within a single database (not common, but can happen, especially with multiple systems sharing a single database).

Table Name	The table name for the core event table. The default is <code>alarm_events</code> .
Event Data Table Name	The table name for event data associated with alarms. The default is <code>alarm_event_data</code> .

Table Definitions

The Alarm Journal system will automatically create the necessary tables for you, and scripting functions can be used to query the system without having to know about the table structure. However, understanding the structure of the Alarm Journal tables can be useful for accessing the data in situations where SQL queries are more convenient.

Alarm Events (`alarm_events`)

This table stores the core data for each event that occurs. An event is a transition for an alarm between active, cleared, or acknowledged. Additionally, other events may be stored in this table that aren't directly related to an alarm, such as a system shutdown event. This table defines a primary key "id", that is used as a foreign key by the Alarm Event Data table, which stores additional information for each event.

Column Name	Data Type	Description
id	integer	A unique integer id for each event entry event
eventid	string	The UUID of the alarm event that this individual event is related to. Each alarm event (one particular active/clear /ack cycle of a defined alarm) receives a unique id in order to distinguish it from other events from the same source.
source	string	The qualified path of the entity that generated the alarm event. See below for more information about qualified paths.
display path	string	The value set for the "Display Path" of the alarm. Generally a user defined, friendlier version of the source.
priority	integer	<p>The priority or severity of the alarm:</p> <ul style="list-style-type: none"> 0: Diagnostic 1: Low 2: Medium 3: High 4: Critical
eventtype	integer	<p>The type of transition represented by this event:</p> <ul style="list-style-type: none"> 0: Active 1: Clear 2: Acknowledgement
eventflags	integer	<p>A numeric bitmask flag field providing additional information about the event.</p> <p>Bit 0: System Event - One of the designated system events. (System Startup, System Shutdown) Bit 1: Shelved Event - The alarm was "shelved" at the time that the event occurred. Shelving alarms does not prevent execution, so if the journal is configured to store shelved events, they will be stored even if they're not sent to the notification system, or shown to users. Bit 2: System Acknowledgement - When the "live event limit" (defined in general alarm settings) is exceeded, the system will automatically acknowledge overflow events, and the acknowledgment event will have this flag set. Bit 3: Acknowledge Event - The event was acknowledged at the time of the event. For events that are cleared after being acknowledged. Bit 4: Cleared Event - The event was cleared at the time of the event. For alarms that are acknowledged after being cleared.</p>
eventtime	datetime	The time of the event.

Alarm Event Data (alarm_event_data)

This table stores the properties associated with an alarm event. The individual event is referenced through the ID column, against the alarm event table.

Column Name	Data Type	Description
id	integer	The id that corresponds to the alarm event in the alarm_events table.
propname	string	The name of the property. May be one of the common alarm properties (a configuration setting), or the name of an associated data property.
dtype	integer	<p>The data type of the property, indicating which data column should be used:</p> <ul style="list-style-type: none"> 0: Integer 1: Float 2: String
intvalue, floatvalue, strvalue	integer, float (double), string	The corresponding value columns for the property. Unused columns will receive "null" values.

Qualified Paths

A qualified path in Ignition is a path to an object, described by various annotated components. Each component has a type identifier and a value, separated by a colon (:), and each component is separated by colon-forward slash (:/). For example, an alarm is identified by alm:Alarm Name. It usually exists under a Tag, in which case, its fuller path would be tag:Path/To/Tag:/alm:Alarm Name. Paths can be built up further depending on the level of specificity required by the situation.

Configuring Alarms

Alarms are conditions that are evaluated when the value of the Tag changes. When the condition becomes true, the alarm is said to be **active**. When it becomes false, the alarm is said to be **clear**. Alarms may also be **acknowledged**. This flags the alarm in Ignition so the acknowledgement state of each alarm can be made visible throughout Ignition. [Acknowledgement](#) also allows users to 'claim' alarms while letting colleagues know that the alarm is being handled. Until an alarm has been acknowledged, it is considered to be **unacknowledged** (unacked).

Alarms can be configured on Tags or OPC items in [SQL Bridge \(Transaction Groups\)](#). The different [Tag types](#) that you can configure alarms on include Memory Tags, Query Tags, Expression Tags, as well as Tags inside of a UDT. You can also put alarms on [System Tags](#) that Ignition inherently provides such as the Gateway Performance, CPU Usage, and more.

Alarm Names and Forward Slashes

Alarm names can make use of forward slashes ("/"). However the alarm name property throughout Ignition is designed to only show characters after the rightmost forward slash. Assume you name an alarm with the following:

```
one/two/three
```

The alarm name property (as seen on the various alarm table components, Tag Browser, and various alarm scripting functions) will omit "one" and "two", leaving the alarm name as:

```
three
```

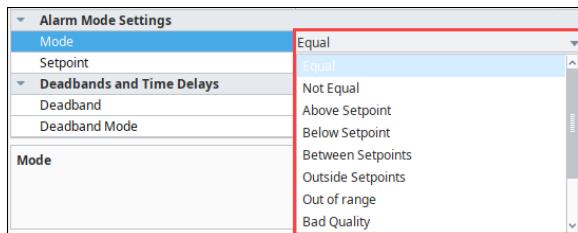
This behavior can be confusing, but is easily avoided by not using forward slashes in alarm names.

Alarm Properties

Similar in concept to properties on Vision components, alarm settings, also known as alarm properties, allow you to modify the behavior of each alarm. Each alarm will have its own alarm properties such as a unique Name, Priority, Display Path, Notes, and many other properties, some of which are optional. Descriptions of each alarm property can be found on the [Tag Properties](#) page.

Alarm Modes and Setpoints

There are many different conditions that can be set up on an alarm. Alarm Mode Settings is where you define the actual conditions when the alarm goes 'true.' Some of the various Alarm Modes are shown in the image below.



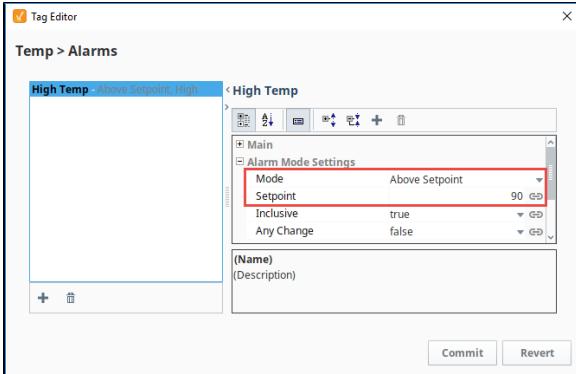
Each alarm is configured with one mode, and usually one or more Setpoints. **Modes** determine the method in which alarm activity is evaluated, while Setpoints are the thresholds or limits that determine when a Tag is within the alarm state. Both properties work in together to determine when alarms become active, as well as when they are cleared.

For example, when Mode is set to "Above Setpoint", the Setpoint attribute is compared to the Tag's value. When the Tag value is above the Setpoint value, the alarm becomes active. Once the value of the Tag is less than the Setpoint value, the alarm will transition to a cleared state.

For a complete list of Alarm Modes and their descriptions, refer to the [Reference Table](#) on the Tag Alarm Properties page.

On this page ...

- [Alarm Names and Forward Slashes](#)
- [Alarm Properties](#)
 - [Alarm Modes and Setpoints](#)
 - [Alarming on Individual Bits](#)
 - [Deadband and Time Delays](#)
 - [Associated Data](#)
- [Configuring Alarms](#)
 - [Configuring an Alarm on a Memory Tag](#)
 - [Configuring Alarms on Tags in a UDT](#)
 - [Configuring Multiple Alarms on a Single Tag](#)
- [Alarm Property Bindings](#)
- [Alarms in Transaction Groups](#)



Alarming on Individual Bits

Additionally, alarms can become active by bit state. A common practice for PLC programming is to store alarm conditions as series of bits inside the PLC and expose them to the world as an integer value. Each bit is essentially a binary value. A series of 8 bits, for example, could be represented as an 8-bit integer. This integer value can be monitored by Ignition's OPC-UA server as an integer value as a Tag. This Tag can then have as many alarms as the integer has bits by monitoring the integer's bit state. Using the Bit Position Mode, multiple alarms in Ignition can be configured to monitor the state of each bit.

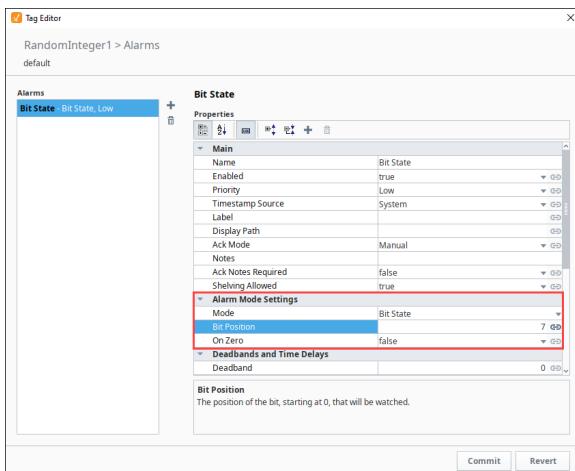
The following table shows binary and decimal equivalents:

Binary	Decimal
10001110	142

In this case, the Tag would be an integer with a value of 142. This Tag would have eight alarms. Each one of these alarms become active in accordance with its Boolean value.

Create Alarms on an Integer Tag Value

1. Edit a Tag that has an integer value.
2. Scroll down to Alarms, and next to 'No alarms,' click on the **Edit** icon to create an alarm.
3. Click the **Add** icon to add a new alarm to the Tag.
4. Fill in the appropriate properties, for example, enter the **Name**, **Priority**, and other properties as appropriate.
5. From the **Mode** section dropdown, select **Bit State** and enter a value for the **Bit Position**.
6. Click **Commit** to save the alarm. Repeat the same process for the remainder of the bits.



Deadband and Time Delays

The value for the deadband is interpreted by the Deadband Mode. All alarms are only evaluated **after** the Tag's value changes, which means that the Tag's own deadband will be considered first. When the deadband is positive, an active alarm condition needs to clear its setpoint(s) by the amount of the deadband for the alarm to clear.

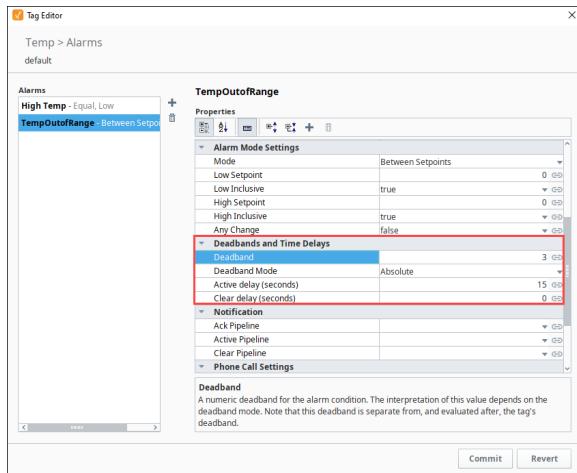


Single Integer – Alarm on Each Bit

[Watch the Video](#)

For example, suppose we have a "Between Setpoints" alarm with a Low Setpoint of 45 and a High Setpoint of 65, and with a Deadband of 3. The alarm will go active if the value is between 45 and 65, but will only clear if the value falls below 42 or rises above 68.

In other situations, a Tag may frequently enter and leave an alarm state, but only for a brief moment. Normally an alarm would be generated each time the alarm condition was met, but the **Active Delay** attribute will prevent any alarms from being generated until the value on the Tag has stayed in an alarm state for a set period of time.



To learn more about deadband and time delays, refer to the [Reference Table](#) on the Tag Alarm Properties page.

Associated Data

You can extend the list of alarm properties by adding your own [Associated Data](#) or custom properties to an alarm you already have configured. These values can be static or dynamic. Static properties are excellent for filtering alarms. Dynamic properties can be driven by another Tag, or evaluated with an expression. Regardless of the type, the Associated Data property values will be attached to the alarm event, viewed in [real-time](#), and recorded in the [Alarm Journal](#) system.

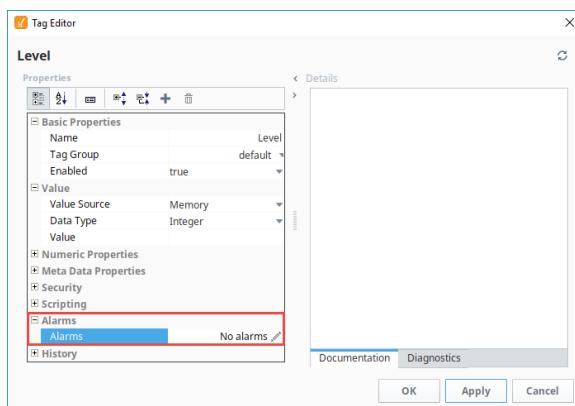
Configuring Alarms

Alarms can be configured on any Tag type: Memory Tag, Query Tag, Expression Tag, as well as Tags inside of a UDT.

Configuring an Alarm on a Memory Tag

Let's use a simple Memory Tag with an Integer data type and create an alarm.

1. In the Tag Browser, double click a memory Tag (i.e. Level) to open the **Tag Editor**.
2. Scroll down to Alarms, and next to '**No alarms**', click on the **Edit** icon to create an alarm, and the alarm UI will slide in from the right.



3. Click the **Add** icon in the bottom left corner of the window, or double click **New Alarm** item to add a new alarm to the Tag.

Active and Clear Delays

[Watch the Video](#)



Configure an Alarm

[Watch the Video](#)

- Configure the following alarm settings:

Name: **Fault Levels**

Priority: **Critical**

Alarm Mode: **Outside Setpoints**

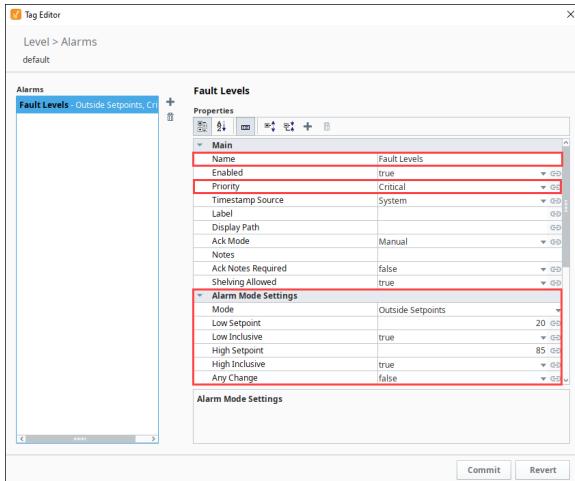
Low Setpoint: **20** (when the Low Setpoint is below 20, an alarm is triggered)

Low Inclusive: **true** (alarm is triggered when the low setpoint is equal to 20 or below)

High Setpoint: **85** (when the High Setpoint is above 85, an alarm is triggered)

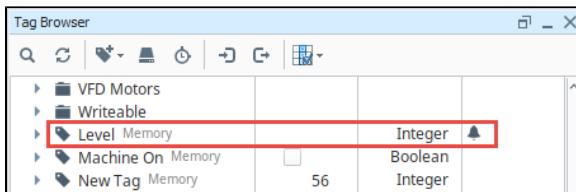
High Inclusive: **true** (when the High Setpoint is equal to or above 85, an alarm is triggered)

- Click **Commit** to save your alarm, or **Revert** to cancel.



- Click **OK** to save your Tag edits.

- Ignition will start monitoring the alarm immediately. Notice that the Alarm icon will appear in the **Traits** column of the Tag Browser next to the name of the Tag. This means that at least one alarm is configured on this Tag.



Configuring Alarms on Tags in a UDT

Alarms can be configured on Tags inside a UDT so when you create instances of that UDT, the alarms will automatically be configured.

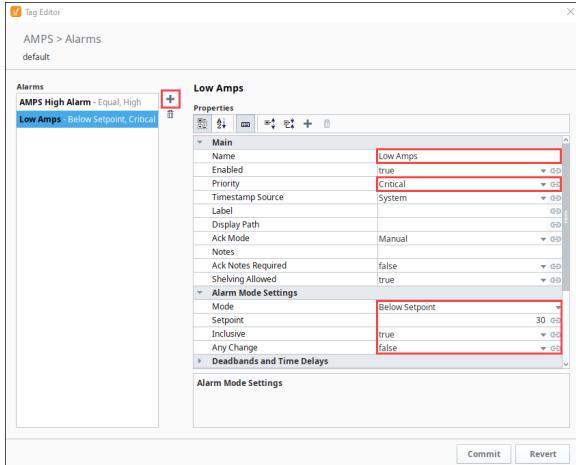
In this example, we have a Motor UDT that contains two Tags: Amps and HOA. Let's configure an alarm on the Amps Tag.

- In the **Tag Browser**, double click on a UDT (i.e., Motor) to open the **Tag Editor**.
- In the Type Structure area, click on a Tag (i.e., Amps).
- Scroll down to Alarms, and next to **No alarms**, click on the **Edit** icon to create an alarm.
- Click the **Add** icon.
- For this example, we entered the following:
Name: **Low Amps**
Priority: **High**
Alarm Mode: **Below Setpoint**
Setpoint: **30**
Inclusive: **true** (an alarm is triggered when the low setpoint is equal to 30 or below)
- Click **Commit** to save your alarm.



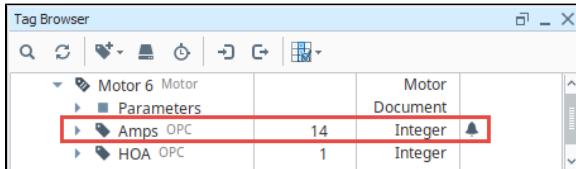
Configure Alarm in UDT

[Watch the Video](#)



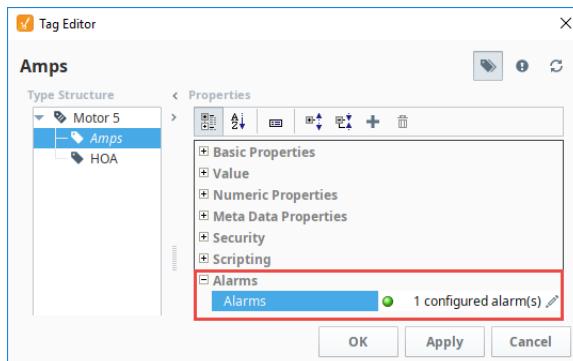
7. Click **OK** to save the UDT. Now you're ready to create instances of the UDT. Once you create your UDT instances, Ignition will start monitoring the alarm immediately.

Note: The Alarm  icon will appear in the **Traits** column of the Tag Browser next to the name of the Tag in the UDT instance. (Notice how you see the name of the Motor UDT next to the Amps Tag of the Motor 6 instance.)



Updates to the UDT and Overriding Alarm Settings

If the alarm in the UDT is updated, the instance will automatically receive the updates and be refreshed. It's not uncommon that you may need to make alarm attribute values unique for particular instances, in which case, you can override alarm settings. Click the override button on the alarm UI of the Tag Editor making it green and edit the alarm properties and/or their values To learn more, refer to Overriding Properties in UDT Instances.



Configuring Multiple Alarms on a Single Tag

Tags can have multiple alarms configured, each with unique setpoints and other configurations. The number of alarms that you can configure is virtually unlimited and will never exceed the bounds of reasonable design consideration. Adding additional alarms is just as easy as adding the first alarm.

Simply click on the **Add +** icon in the alarm UI and start configuring the new alarm. This way separate alarms can monitor different setpoints, so Tags can have alarms for high setpoints and low setpoints.



Multiple Alarms on a Tag

Note: Alarms can be copied and pasted with keyboard shortcuts. Select an alarm from the Tag Editor, press **Ctrl-C** to copy, and **Ctrl-V** to paste. Alarms with similar configurations can rapidly be replicated to hasten development.

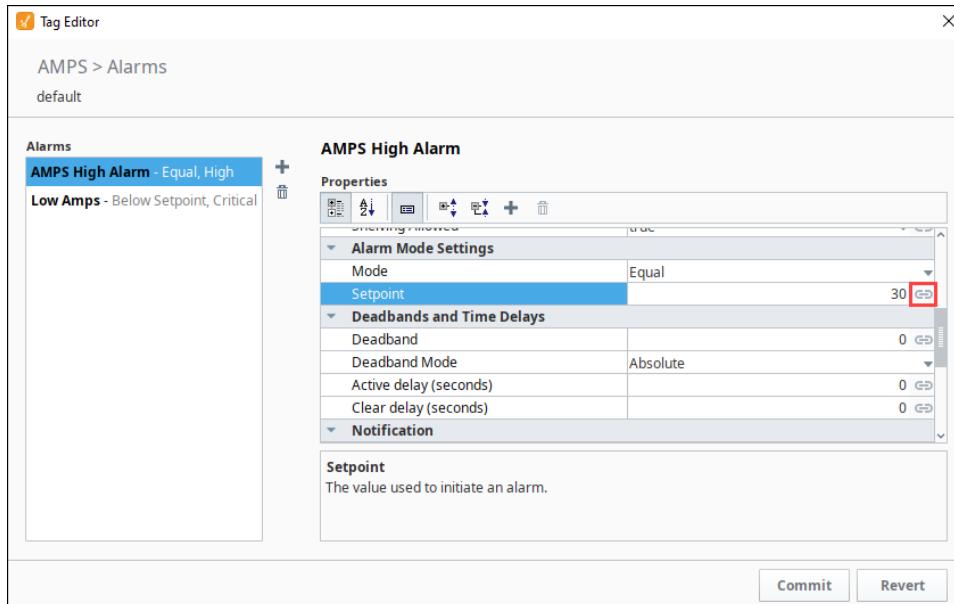
[Watch the Video](#)

Alarm Property Bindings

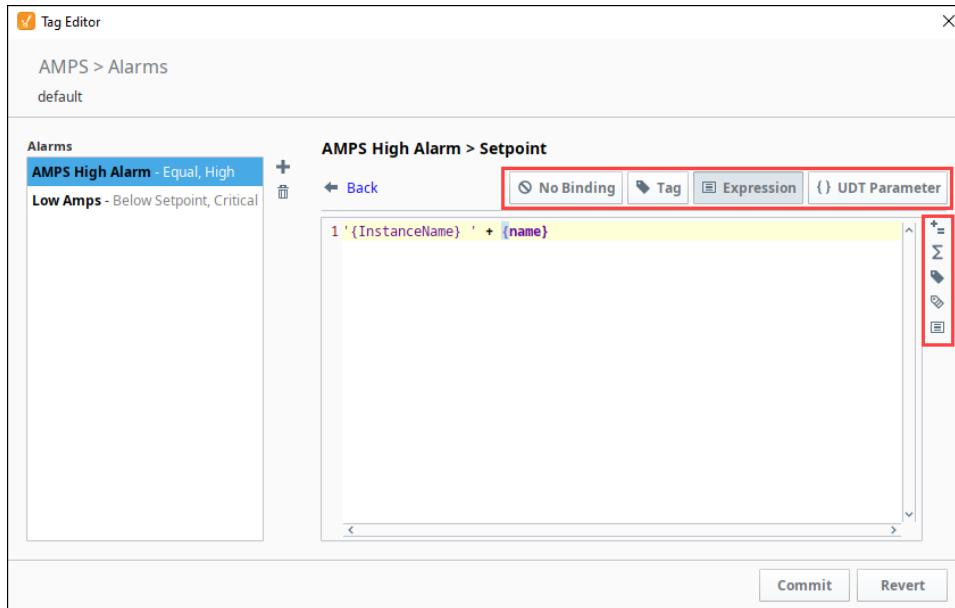
Many alarm properties are bindable, which means they can be bound to other Tags in the system, expressions and even a UDT parameter. For example, you might bind the Enabled property to another Tag which represents whether or not your process is running, thereby disabling the alarm when production is stopped. Or, you might bind the setpoint of an alarm to a Tag that operators can manipulate, thereby letting the setpoint be changed at runtime.

This example shows how to bind an alarm property from the Tag Browser in Perspective.

1. Double click on your Tag to open the **Tag Editor**.
2. Click on the **Edit** icon next to **Configured Alarms** to see your configured alarms. (If you don't have any alarms, create an alarm using the steps in the [Configuring an Alarm on a Tag](#).)
3. Select the alarm and the screen will refresh with all the alarm properties.
4. From here, find the alarm property you want to add a binding to and click on the binding icon. The binding UI will slide in from the right.



5. Select the binding type (**No Binding**, **Tag**, **Expression**, or **UDT Parameter**, if applicable). The image below shows an example of an Expression binding. Notice that the expression can reference many useful values such as the Tag's value and other settings of the alarm. Enter your expression.
6. Once you configured the binding to your liking, click **Commit**, or **Revert** if you decide to cancel.



7. Click **OK** to save the changes to the Tag.

For more information on property bindings see, [Property Bindings in Perspective](#) and [Property Bindings in Vision](#).

Alarms in Transaction Groups

Alarms can also be added to OPC items in Transaction Groups. This means alarms can be used without ever creating a Tag in Ignition. Simply edit an OPC item, and an Alarming section will appear in the Tag Editor window. From here, adding an alarm to the item is similar to adding an alarm to an Ignition Tag.

Value Source	OPC
Data Type	Integer
OPC Server	Ignition OPC UA Server
OPC Item Path	ns=1;s=[Dairy]_Meta:Overview/AU 1/Fan 2 HOA

Related Topics ...

- [Tag Alarm Properties](#)
- [Tag Properties](#)
- [Alarms in UDTs](#)
- [Alarm Associated Data](#)
- [Vision Alarm Status - Acknowledgement](#)
- [Property Bindings in Perspective](#)
- [Property Bindings in Vision](#)

In This Section ...

Dynamic Alarm Attributes

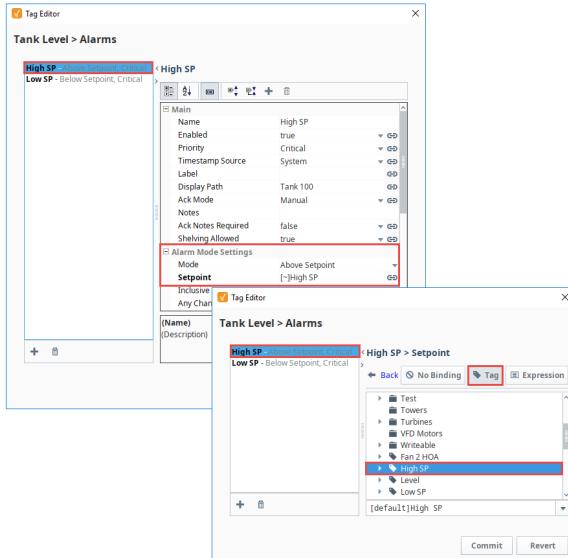
Dynamic Setpoints

You can configure alarms with dynamic setpoints inside of a Tag definition. It is similar to configuring an alarm, however, configuring an alarm with a dynamic setpoint requires additional Tags to serve as these setpoints. A good example of this is when an operator changes a high or low setpoint, it also changes how the alarm is evaluated.

To Configure Dynamic Setpoints on an Alarm

Suppose you want to alarm a Tank Level if it goes above or below a certain setpoint. Rather than hardcoding a value in the high and low Setpoint properties, you can bind them to either a Tag in the system or an Expression to make them dynamic. This example uses an OPC Tag called Tank Level, and two Memory Tags called High SP and Low SP. The Tank Level will use the values in the Memory Tags as the high and low setpoints. Let's alarm the tank level when the setpoint goes above 85 and below 20.

1. Create two Memory Tags calling one **High SP** with value of 85, and another one called **Low SP** with a value of 20.
2. Use an OPC Tag and rename it to **Tank Level**. Click on the Tag to open the **Tag Editor**.
3. Scroll down to Alarms, and click on the **Edit** icon next to your alarm to open the Alarm UI.
4. If you don't have a configured alarm, you will need to create one. Select the configured alarm that you want associate a high setpoint with. (This examples uses the **High SP** alarm).
5. Under Alarm Mode Settings, set the Mode to **Above Setpoint**.
6. Click on the **binding** icon for the **Setpoint** property. You can select a **Tag** or **Expression**. (This examples binds the **Setpoint** property to a Memory Tag that was created called **High SP**). Click the **Commit** to commit your changes.



7. Now let's configure the low setpoint. Create another alarm to associate with the low setpoint if you don't have one.
8. Under Alarm Mode Settings, set the Mode to **Below Setpoint**.
9. Click on the **binding** icon for the **Setpoint** property. You can select a **Tag** or **Expression**. (This examples binds the **Setpoint** property to a Memory Tag that was created called **Low SP**). Click the **Commit** to commit your changes.
10. Click **OK**. Now, the tank alarm will be evaluated based upon the high and low setpoints of each Tag.

On this page ...

- [Dynamic Setpoints](#)
 - [To Configure Dynamic Setpoints on an Alarm](#)
- [Dynamic Enabling and Disabling](#)
 - [Enabling Based on Machine State](#)
 - [Enabling Based on Time of Day](#)



UDT Alarm Dynamic Setpoints

[Watch the Video](#)

11. To test it, change the value of the Tank Level to 99, and you'll notice it activates the alarm.

Tag Browser			
- Tank Level OPC	99	Float	▲
Enabled	<input checked="" type="checkbox"/>	Boolean	
OpcItemPath	ns=1;s=[Generic]_Meta:Writeable...	String	
OpcServer	Ignition OPC UA Server	String	
Quality	Good	String	
TagGroup	Default	String	
Timestamp	2019-06-03 12:19:55 PM	DateTi...	
value	99	Float	
Alarms		String	
High SP		DateTi...	
AckTime		String	
AckUser		String	
AckUserName		DateTi...	
ActiveTime	2019-06-03 12:19:56 PM	DateTi...	
ClearTime		DateTi...	
DisplayPath	Tank 100	String	
DisplayPathOrSource	Tank 100	String	
EventState	Active	Integer	
EventTime	2019-06-03 12:19:56 PM	DateTi...	
EventValue	99.0	String	
IsAcked	<input type="checkbox"/>	Boolean	
IsActive	<input checked="" type="checkbox"/>	Boolean	
IsClear	<input type="checkbox"/>	Boolean	
IsShelved		Boolean	
Label		String	
Name	High SP	String	
Priority	Critical	Integer	
Source	prov:default/tag:Tank Level/al...	String	
State	Active, Unacknowledged	Integer	
Low SP		String	



While you can configure dynamic values on any property that is showing the binding icon on the right, some other properties (like Name) may also accept dynamic properties using the {myParam} notation. We recommend against using Dynamic setpoints on these static alarm properties because they will only evaluate on startup, not while the Gateway is running.

Dynamic Enabling and Disabling

Enabling Based on Machine State

Allowing a dynamic condition to determine if an alarm is enabled or disabled is possible inside the alarm's properties, like whether or not a machine is on. This example uses two Tags: one for the Alarm, and one to determine the running state of the machine. Two conditions must be true for the alarm to become active. The value on the alarm Tag must match the setpoint ('true' in the example below), and the Enabled property must evaluate to 'true' driven by the value of the 'Machine On' Tag.

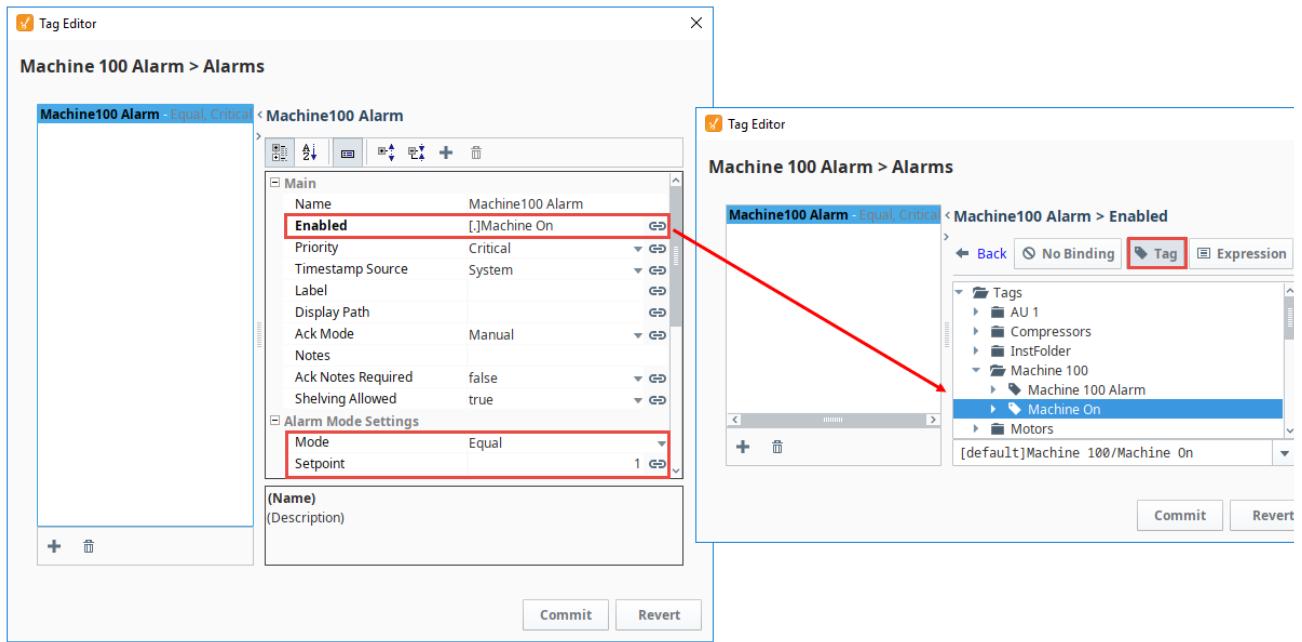
1. Select a **Tag** that has the alarm you want to configure.
2. To access the alarm properties, click on the **Edit** icon next to your alarm to open the Alarm UI. Select an alarm or create a alarm if one doesn't exist. The alarm in this example is called Machine 100 Alarm.



Enabling Alarms Dynamically – Machine On

[Watch the Video](#)

3. Click the **binding** icon for the **Enabled** property and bind it to a **Boolean Tag** or an **Expression** that evaluates a 'true' or 'false' condition. This example uses a Tag called Machine On. Enter the **Mode** to '**Equal**' and the **Setpoint** to '**1**'. Click **Commit**.



4. Click **Commit** to save your changes. Now, the alarm will only be evaluated based upon the state of the Machine On Tag. Set the Machine to **On** or 'true.'

5. In the Tag Browser, expand your Tag (i.e., Machine 100 Alarm), and you'll notice the '**IsAlert**' is now active. As you can see, making the Enable property dynamic based on another Tag in the system is extremely easy.

Tag Browser

Machine 100				
Machine 100 Alarm OPC				
Enabled	<input checked="" type="checkbox"/>		Boolean	
OpcItemPath	<input checked="" type="checkbox"/>	ns=1;s=[Generic]_Meta:Writeable/Writeabl...	String	
OpcServer		Ignition OPC UA Server	String	
Quality		Good	String	
TagGroup		Default	String	
Timestamp		2019-06-03 2:33:13 PM	DateTime	
value	<input checked="" type="checkbox"/>		Boolean	
Alarms				
Machine100 Alarm				
AckTime			String	
AckUser			String	
AckUserName			String	
ActiveTime		2019-06-03 3:43:18 PM	DateTime	
ClearTime			DateTime	
DisplayPath			String	
DisplayPathOrSource		Machine 100/Machine 100 Alarm/Machine...	String	
EventState		Active	Integer	
EventTime		2019-06-03 3:43:18 PM	DateTime	
EventValue		true	String	
IsAcked	<input type="checkbox"/>		Boolean	
IsActive	<input checked="" type="checkbox"/>		Boolean	
IsClear	<input type="checkbox"/>		Boolean	
IsShelved	<input type="checkbox"/>		Boolean	
Label			String	
Name			String	
Priority		Critical	Integer	
Source		prov:default:/tag:Machine 100/Machine 10...	String	
State		Active, Unacknowledged	Integer	
Machine On Memory		<input checked="" type="checkbox"/>	Boolean	

Enabling Based on Time of Day

There are occasions when you only want alarms to be evaluated at certain times of the day. In Ignition, you can automatically enable and disable alarms for specific times of the day. This is typically achieved by binding the alarm's Enabled property to a Tag or an Expression. (This example uses an Expression).

1. From the **Tag Browser**, select the **Tag** on which you want to configure the alarm.
2. To access the alarm properties, click on the **Edit** icon next to your alarm to open the Alarm UI.
3. Select an alarm you want to place the binding on, or create a alarm if one doesn't exist. The alarm in this example is called Machine 100 Alarm.
4. Bind the **Enabled** property to an expression that evaluates a 'true' or 'false' condition based upon the current time. Click the binding button to the right of the Enabled property, and click on **Expression** tab on the top right of the screen, and enter your expression or copy and paste from the code block below, then click **Commit**.

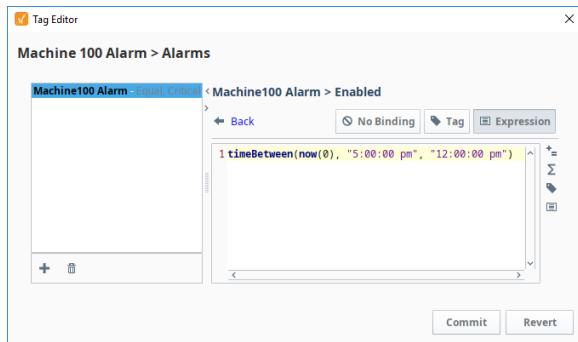
In the expression language, there are various functions for dates that can be used. For example, the following expression would return 'true' if the time is between the hours of 5pm and 12pm, and return 'false' if it is not.



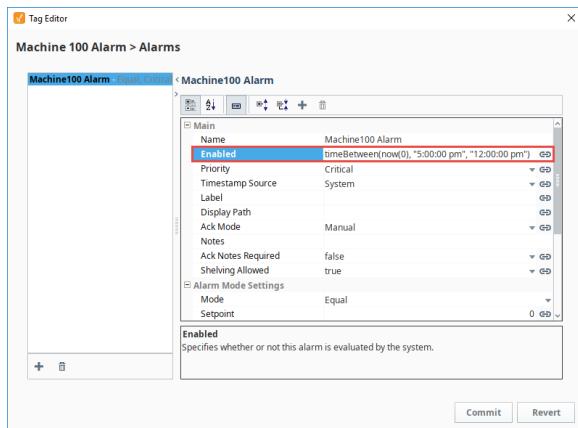
Enabling Alarms Dynamically – Time of Day

[Watch the Video](#)

```
timeBetween(now(0), "5:00:00 pm", "12:00:00 pm")
```



5. Click **Commit** again to save your edits. The alarm will only be evaluated when the system clock falls between the specified time.



Related Topics ...

- [Alarms in UDTs](#)
- [Alarm Associated Data](#)
- [Alarm Event Properties Reference](#)

Alarms in UDTs

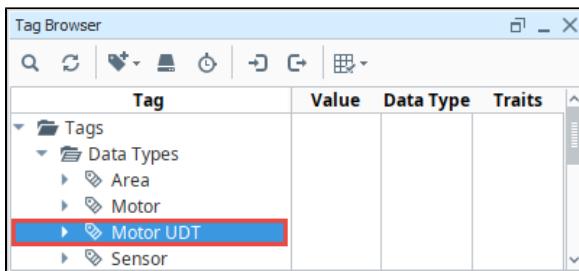
Configuring an Alarm on a UDT Member

The great thing about UDTs is that you configure it in one place, inside of the UDT definition, and every instance of that UDT will automatically inherit that same configuration. The same concept works for alarms on a UDT. If an alarm is configured inside a UDT, every instance of that UDT will automatically have that same alarm configuration. Even if a new instance is created, it will automatically get that same alarm configuration.

Configuring an Alarm on a UDT

This example uses a Motor UDT. The Motor UDT contains two OPC Tags: Amps and HOA. Let's configure an alarm on the Motor UDT when the Amps setpoint goes under 25.

1. In the **Tag Browser**, go to the **Data Types** folder, and double click on a UDT (i.e., Motor UDT) to edit the definition.



2. Select a Tag (i.e., Amps) and scroll down to Alarms, and click on the **Edit** icon next to your alarm to open the Alarm UI.
3. Click on the **Add** icon on the lower left side of the screen to create a new alarm. Enter an **Alarm Name**, **Display Path**, **Priority**, **Mode**, and **Setpoint**. UDT configurations are setup the same way as the normal [alarm configuration on a Tag](#).

Display Path

Since you have multiple instances of a Motor, as in this example, you need to make sure that the Display Path is unique for every instance, otherwise, every instance of the UDT will have the same Display Path name, and the operator won't know which Motor alarm to respond to. There are a couple of ways to address this. You can leave the Display Path blank, in which case, Ignition will use the full Display Path to the instance of that Tag, or you can bind a property to an expression involving a UDT parameter (i.e., "Motor" + {MotorNumber}).

4. Click **Commit** to save your alarm edits.
5. Click **OK** to save your Motor UDT edits. Now, all instances of the UDT will have a similar alarm setup, but each alarm can be unique to that UDT instance by changing the **Below Setpoint** value.

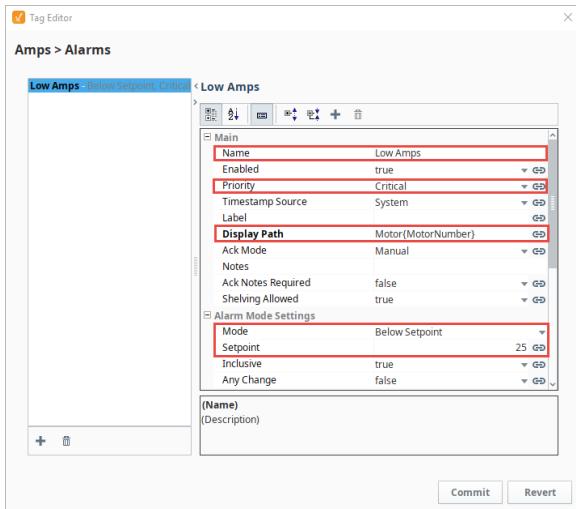
On this page ...

- Configuring an Alarm on a UDT Member
 - Configuring an Alarm on a UDT
- Dynamic Setpoints in UDTs
 - Configuring Dynamic Setpoints Inside a UDT



Configure Alarm in UDT

[Watch the Video](#)



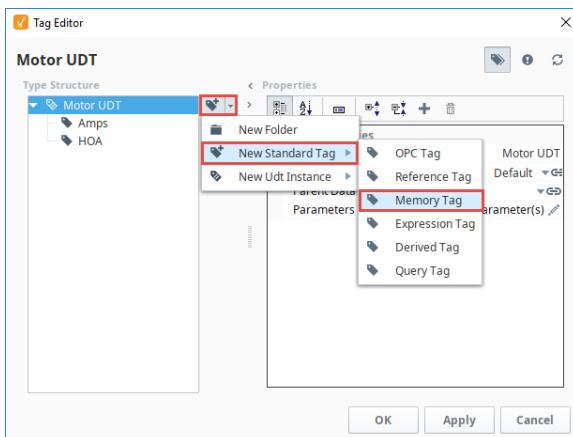
Dynamic Setpoints in UDTs

Instead of using a hardcoded setpoint, you can configure alarms with dynamic setpoints inside of a UDT definition. It is similar to configuring a UDT alarm, however, configuring a UDT alarm with a dynamic setpoint requires additional Tags to serve as these setpoints in all the deployed UDTs.

Configuring Dynamic Setpoints Inside a UDT

In this example, we'll use the Motor UDT in the above section to create a **Memory Tag** inside the UDT definition to serve as the setpoint.

1. In the **Tag Browser**, go to the **Data Types** folder, and click on your UDT to edit the definition.
2. Click the **Add Tag** icon and select **New Standard Tag > Memory Tag** from the dropdown.

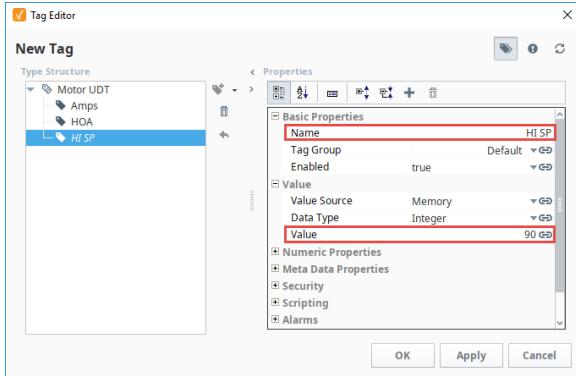


3. Enter the **Name** of the Tag (i.e., HI SP) and a **Value** (i.e., 90). Click **Apply**.

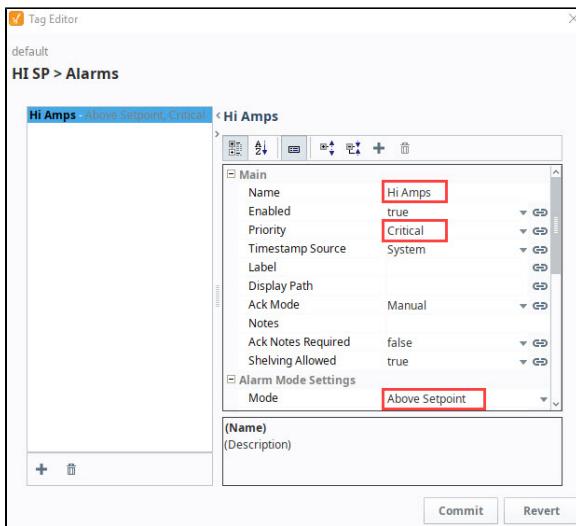


UDT Alarm Dynamic Setpoints

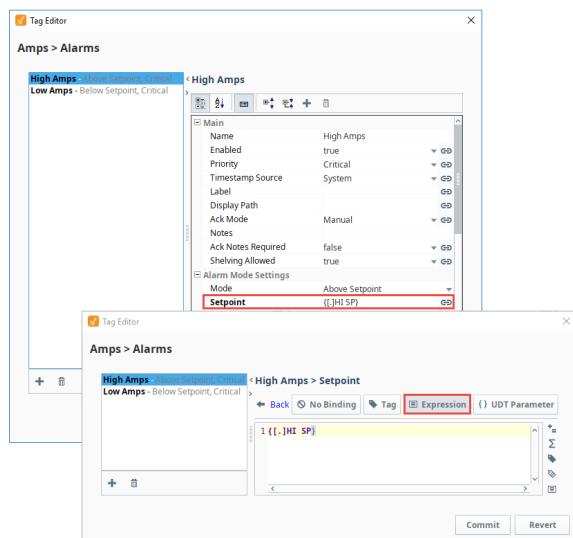
[Watch the Video](#)



4. In the **Type Structure** area, select the Tag you want to associate with the dynamic setpoint (i.e., Amps)
5. Under Properties, scroll down to Alarms. Click on the **Edit** icon next to Alarms.
6. Click on the Add icon to create a new alarm for the high setpoint and call it "Hi Amps".
7. Set the **Priority** as **Critical**, and the **Alarm Mode** as **Above Setpoint**. Click **Commit**.



8. Bind the **Setpoint** property to the new UDT Memory Tag (i.e., HI SP). Click on the **binding** icon and you can either create an expression or use the UDT Tag Tab to select the Memory Tag (i.e., HI SP). This example uses an expression as shown in the image below.
9. Click **Commit** to save the expression.
10. Click **Commit** again to save your alarm edits.



11. Click **OK** to save all your UDT updates. Now you're ready to create instances of your UDT. All of the UDT instances will now have Memory Tags that serve as alarm setpoints.
12. Test it out by dragging an Alarm Status Table component in a window and locating your alarm.

Related Topics ...

- [Configuring Alarms](#)
- [Dynamic Alarm Attributes](#)
- [UDT Multi-Instance Wizard](#)

Alarm Associated Data

What is Alarm Associated Data

Every alarm in Ignition has alarm properties associated with it as it moves throughout the system, like active time, clear time, who acknowledged the alarm, time the alarm was acknowledged, priority, display path, and much more. You can extend the list of alarm properties by adding your own associated data or custom properties to an alarm you already configured. Values can either be static or dynamic. Static properties are great for filtering alarms. Dynamic properties will often be bound to other Tags or an expression. Regardless of the whether a value is static or dynamic, these values will be attached to the alarm event as it moves through the system, and the values will be available from the [Alarm Status](#) system, the [Alarm Journal](#) system, and the [Alarm Notification](#) system.

It's easy to go back to an existing alarm and add associated data.

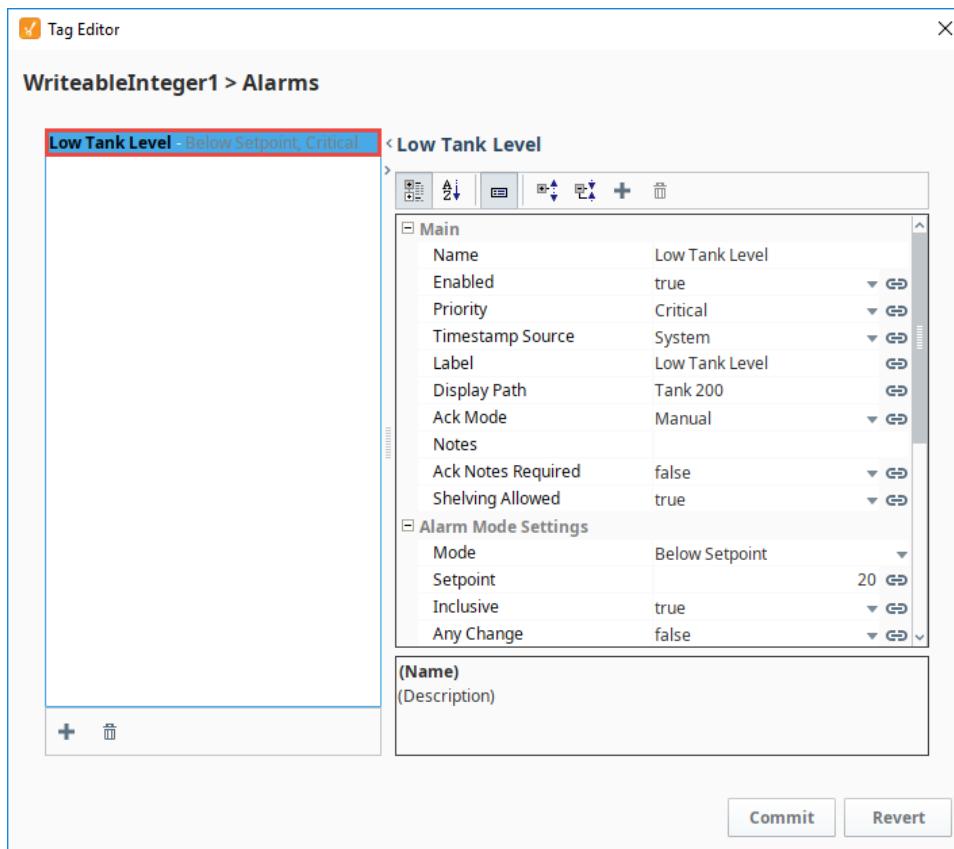
On this page ...

- [What is Alarm Associated Data](#)
 - [Creating Associated Data for both Static and Dynamic Properties](#)
- [Alarm Grouping](#)
 - [Creating an Alarm Grouping](#)

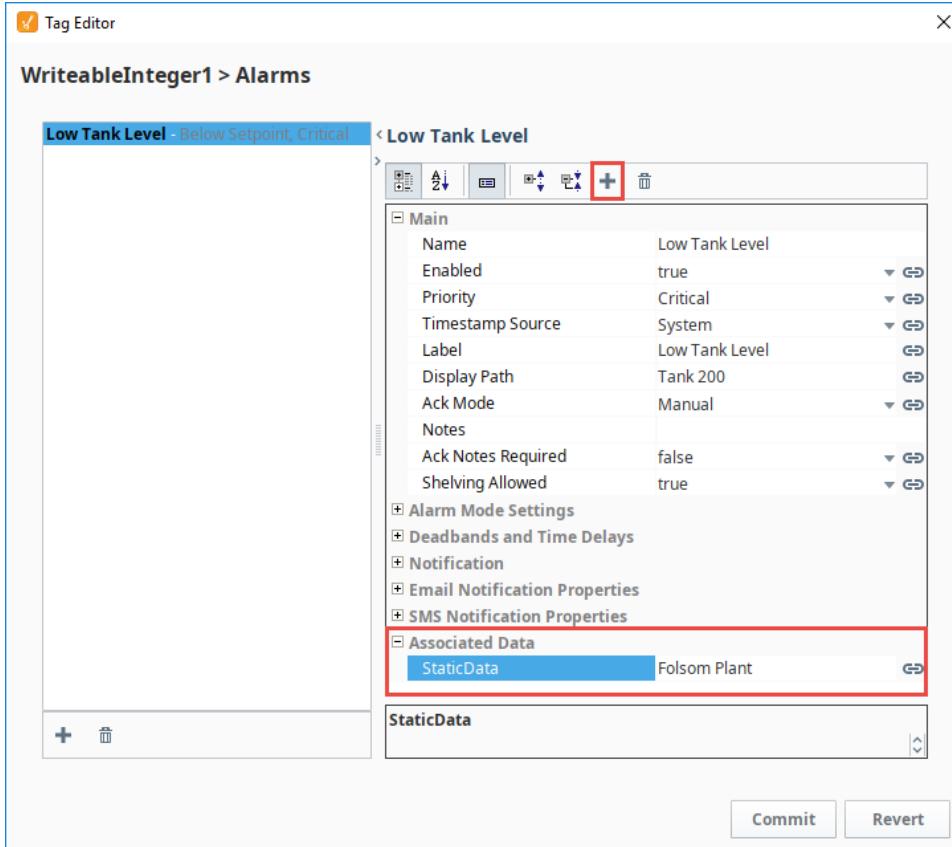
Creating Associated Data for both Static and Dynamic Properties

For this example you can use an existing alarm you already configured or [create a new alarm](#) if you don't have one.

1. Right-click on your Tag and select **Edit Tag**. This example uses a WriteableInteger OPC Tag, but any alarm-capable Tag will suffice.
2. In the **Tag Editor**, scroll down to Alarms, and next to 'No alarms,' click on the **Edit**  icon and click on your alarm. The Tag below already has an alarm named 'Low Tank Level.' Select the alarm to open the Alarm UI.



3. Click the **Add**  icon above the alarm properties to add new associated data for the alarm. Scroll to the bottom of the property list, and you'll see a new associated data property was added. By default, the new property name is called '**New Data**' Double-click the property name to rename it to something more meaningful. In this example, make the associated data property a static value. We renamed ours to '**StaticData**' with a value of '**Folsom Plant**' and press enter to submit.



4. Now, let's create another Associated Data property with a dynamic value. Click the **Add**  icon above the alarm properties.
5. Scroll down to the bottom of the property list and rename your 'New Data' property to '**DynamicData**' and bind it to a Tag using the **binding**  icon on the right side of the window. This example uses a Ramp0 OPC Tag.
6. Click the '**Commit**' to submit the Tag selection, and then click **OK** to save your Tag edits.

7. When the alarm goes active, Ignition stores the values of the associated data with the alarm. You can view the details of the alarm in the [Alarm Status Component](#) along with the new associated data properties. To view the **Details** of the alarm, check the box of the alarm you want to

see, and click on the magnifying glass at the bottom of the window to open the Details window.

The screenshot shows the Ignition Alarm Details window. At the top, there is a table with columns: Active Time, Display Path, Priority, Current State, and Label. One row is selected, highlighted in red, with the following values: Active Time (6/7/19, 11:46 AM), Display Path (Tank 200), Priority (Critical), Current State (Active, Unacknowledged), and Label (Low Tank Level). Below the table is a 'Details' tab where various alarm properties are listed. Some properties have their values highlighted with a red box, specifically 'DynamicData' (995.093333333332), 'Group' (Production), and 'DynamicData' again ([-]Ramp/Ramp0). At the bottom of the window are 'Acknowledge' and 'Shelve' buttons, along with some icons.

Alarm Grouping

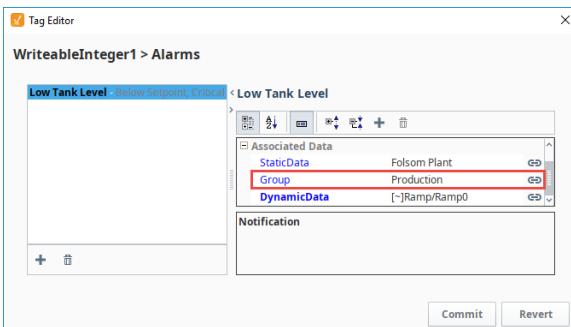
Grouping alarms in Ignition is an important concept. Instead of seeing every alarm in a single view, often times, you may want to view alarms for a particular area of your plant, or for a specific set of alarms.

There are several ways to group alarms. One way is to use a folder structure which requires you to organize your Tags into a hierarchy. Another way is to use the Display Path field in the alarm configuration. The third way, and the most recommended way, is to use associated data. It's a common design practice to associate alarm groupings on the associated data of the alarm.

Creating an Alarm Grouping

This example uses the WriteableInteger Tag that we used in the example above.

1. Use an existing alarm that you already configured. If you don't have one, [create a new alarm](#).
2. Right-click on your **Tag** and select **Edit Tags**. In the **Tag Editor**, scroll down to Alarms, and next to 'No alarms,' click on the **Edit** icon and click on your alarm. Click the **Add** icon above the alarm properties to add new associated data for the alarm.
3. Scroll to the bottom of the property list, and you'll see a new associated data property was added. In this example, make the associated data property a static value, and rename '**New Data**' to '**Group**' with a value that represents your Group (i.e., Production).



4. Click **Commit** to save your alarm edits, and then press **OK**. The alarm will now always have the '**Group**' designation associated with it. This can be used for filtering or part of your alarm pipeline notifications.
5. Before you can use the Alarm Journal to filter on your new associated data property, edit the '**filterAlarm**' Extension Function to include your new associated data property by selecting the **Alarm Journal component**, and right clicking on **Scripting**. Under the **Extension Functions** folder, click on **filterAlarm**, then click on the **Enabled** checkbox to enable the script. Edit the script to add your new property and property value. You can copy the new code below to add



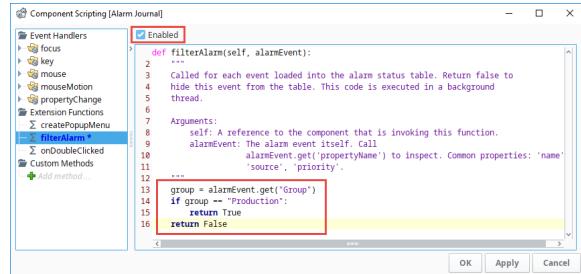
Alarm Grouping

[Watch the Video](#)

to your filterAlarm script.

filterAlarm scripting function

```
group = alarmEvent.get("Group")
if group == "Production":
    return True
return False
```



6. Now, you're good to go! Open the [Alarm Journal](#) component and filter on your associated data. The Alarm Journal Table shown below displays all the alarms that are associated with the group, Production.

Event Time	Display Path	Event State	Priority	Event Value	Current State	Label
6/7/19, 2:35 PM	Pressure/HighPressure	Active	Critical	99	Active, Unacknowledged	HighPressure
6/7/19, 2:35 PM	Pressure/HighPressure	Active	Critical	99	Active, Unacknowledged	HighPressure
6/7/19, 2:30 PM	Tank 200	Active	Critical	7	Active, Unacknowledged	LowTank Level
6/7/19, 2:30 PM	Tank 200	Active	Critical	7	Active, Unacknowledged	LowTank Level
6/7/19, 2:30 PM	Tank 200	Clear	Critical	55	Cleared, Unacknowledged	LowTank Level
6/7/19, 2:30 PM	Tank 200	Clear	Critical	55	Cleared, Unacknowledged	LowTank Level
6/7/19, 12:42 PM	Tank 200	Active	Critical	3	Active, Unacknowledged	LowTank Level
6/7/19, 12:42 PM	Tank 200	Active	Critical	3	Active, Unacknowledged	LowTank Level

8 events

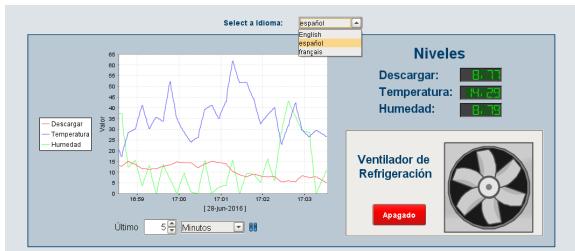
Related Topics ...

- [Configuring Alarms](#)
- [Extension Functions](#)
- [Alarm Status Table](#)
- [Alarm Journal Table](#)

Localization and Languages

Translating Ignition

Localization allows you to translate text into multiple languages in a project for display on client screens. The localization feature allows users located in different countries to set their default language so client screens can be displayed in their native language. The user can easily choose which language their Client displays with a click of a button! Text contained in components, that have their properties correctly mapped with localization, will change to reflect the language the user has selected. There is a special [Language Selector](#) dropdown list that displays the available translation mapping options for these translations.

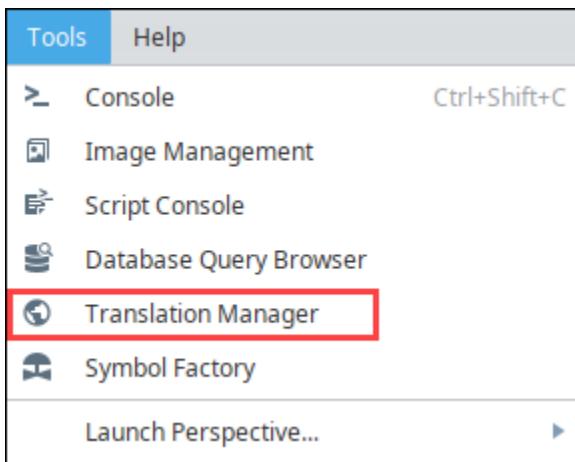


On this page ...

- Translating Ignition
- Translation Manager
- Translation Database and Term Lookup
 - Quickly Identify Fields to Translatable Terms
- Previewing Translations

Translation Manager

All translations are stored centrally in the Gateway and are distributed to each Client and Designer. All projects share the same translations, and those translations can be used in other locations, such as Gateway scripts, and alarm messages. The full translation database can be viewed and edited in the **Translation Manager** tool located on the Tools menu the Designer.



You can add new terms and phrases to the Translation Manager. Make your phrases as long or short as you like with single words or whole paragraphs. It allows users to easily [import and export](#) translatable components and share them across projects and with other users. It also has the ability to define new languages.

A central term database and support are automatically built in to all component text properties, as well as other text-based properties, such as alarm messages. Aside from defining translations for terms, there is usually no other work needed to take advantage of the translation system.

Translation Manager			
Languages	Translation Terms		
(All)	Key	English (Alternate)	Spanish
<input checked="" type="checkbox"/> English	+ Auto		Automatico
<input checked="" type="checkbox"/> Spanish	+ brake		freno
	+ Brake Motors		Motores de Freno
	+ Days		Dias
	+ Hand		Manual
	+ Humidity		Humedad
	+ Last		Ultimo
	+ Minutes		Minutos
	+ Off		Apagado
	+ Seconds		Segundos
	+ Tank		Deposito
	+ Temperature		Temperatura
	+ Value		Valor

Translation Database and Term Lookup

Terms are referenced in the translation database using direct string comparison. Ignition has special term **Keys** that are used to identify when to translate text. The base term may be a user-readable string such as "Start", or any special code you want such as "START_COMMAND" or "#start", which would have an alternate translation for English defined as "Start." This way you can quickly identify which terms will be translated when working in the Designer. To keep things simple, the English key is automatically used if no match is found for the requested language.

Quickly Identify Fields to Translatable Terms

You can quickly and easily identify, and translate all terms by selecting a container from your Project Browser and opening the Translatable Terms panel.

Note: If you don't see the Translatable Terms panel, go to the menubar and select **View > Panels** and check **Translatable Terms**. Here, you can see all terms for each component in the container, and if each term has an associated translation for the requested language.

Translatable Terms

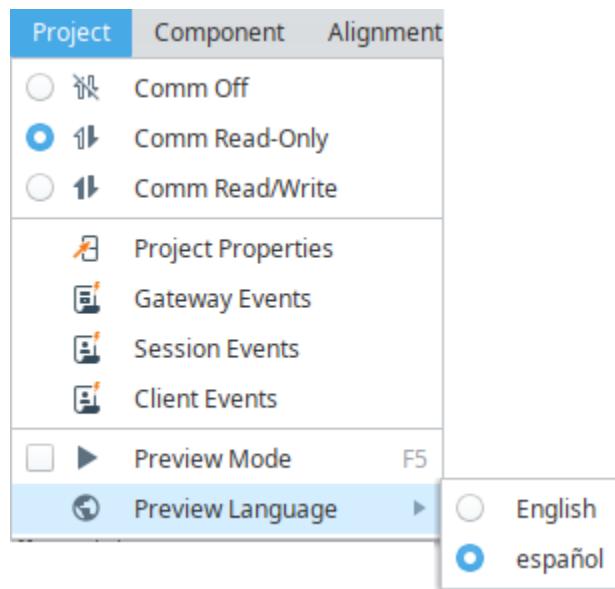
The dialog box has a toolbar with icons for file operations and search. Below the toolbar is a section titled "Component Terms". A table lists terms with their keys and Spanish translations:

Description	Key	Spanish (Com...)	Spanish (Glob...)
Confirm Text	Are you sure?		
selectedText.1	Hand		Manual
unselectedTex...	Hand		Manual
selectedText.2	Off		Apagado
unselectedTex...	Off		Apagado
selectedText.3	Auto		Automatico
unselectedTex...	Auto		Automatico
Confirm	Confirm		

Selected Component: Multi-State Button

Previewing Translations

You can easily preview your translations in the Designer. To change the preferred language for Preview Mode, use the **Preview Language** menu item under the **Project** menu in the Designer. When you put the Designer into Preview Mode, everything on your windows will be automatically translated. Using the Language Selector Component or scripting can temporarily change the current language, and the language will revert back any time Preview Mode is disabled. The Designer will remember your choice, even when the Designer is restarted.



In This Section ...

Creating Translation Lists

Project Translation Overview

Project translation is also known as Client localization. This is how the user can choose their language-of-choice for the project they are working with. When the user selects a language, the Client will change its text in accordance with the project's translation settings. Translations work with all components (i.e., Labels, Charts, or Multi-State Buttons).



Translation Notes

- Text within Vision Table components (those under the "Table" group) can not be translated. However the Vision Alarm Status Table and Vision Alarm Journal Table components do support translations.
- For Text elements that have bindings, the translation occurs after the binding is evaluated.

The Translation Manager allows you to view all the terms from all your projects in the global translation database. You can add, edit, and remove terms, and provide translations, as well as allow users to easily import and export translatable components. The translation database shares terms across all projects and with other users.

Setting up New Languages, Terms, and Translations

Setting up new languages and adding terms for translation is pretty easy. For every element of text on the screen, you can provide a translation in the desired language. When a user logs into a project, they can choose their preferred language, and all the text will be displayed in their preferred language. They can also have their preferred language defined in [user settings](#) so when a user logs into the Client, their preferred language is automatically enabled.

Component vs Global Terms

There are two types of terms used in the translation system: Component and Global. Both behave a little differently, and it's important to know the difference.

- **Component Terms** - Component level terms are specific to that one component, and translations are added to the translation system using the Translatable Terms Panel. Any component level translation will automatically take precedence in the event there is a matching global term.
- **Global Terms** - Global terms and translations are available in both the Translatable Terms Panel and in the Translation Manager's global database. The global translation database provides language translations on all components, in all windows, and on all projects. You can setup global translations so that anywhere a term is used within your project, it will get replaced with that global translation. Although, there are times, when a component might have a different context than the global translation. For this reason, you may want to create a component level translation to take precedence over the global translation. For example, some users may work in a specific window of a project, and what is typically called a 'Tank,' this group of users refer to it as a 'Barrel.' You can create a component level translation for 'Barrel' to take precedence on the root container of a window. This way the users see the term in the context that is familiar to them.

Add a New Language

1. In Designer, right click in your workspace and select **Translations** from the dropdown. The Translatable Terms window will open.
2. Select the icon, and click **Add Language**. By default, the English language is defined.

On this page ...

- [Project Translation Overview](#)
- [Setting up New Languages, Terms, and Translations](#)
 - [Component vs Global Terms](#)
 - [Add a New Language](#)
 - [Add Global Translation Terms](#)
 - [Add Component Level Translation Terms](#)
- [Translation List Import and Export](#)
 - [Export Terms from a Window](#)
 - [Import Terms](#)



INDUCTIVE
UNIVERSITY

Introduction to Project Translation

[Watch the Video](#)

The screenshot shows the 'Translatable Terms' window. At the top, there's a toolbar with icons for file operations and search. Below it is a component selection dropdown where 'Spanish' is selected. A red box highlights the 'Add Language' button next to the dropdown. The main area is a grid table with columns: Component, Description, Spanish (Component), and Spanish (Global). The table contains several entries, such as 'Label.Text' with 'Motors' and 'Motores de Freno', and 'Multi-State Button' with various states like 'Hand', 'Off', 'Auto', and 'Confirm'. The bottom of the window shows the selected component as 'Root Container' and has tabs for 'Properties' and 'Translations'.

3. Select a new language from the dropdown.

The screenshot shows the 'Add new language' dialog box. It has a dropdown menu at the top with 'French' selected. A red box highlights the 'French' entry in the list below. Other languages listed include Finnish, Friulian, Fulah, Galician, Ganda, and Georgian.

Add Global Translation Terms

Creating global translations is always a good first choice over the component level translation because most of your translated terms will be used across all projects unless you want one particular term to be different from the global translated term.

There are a couple of ways to add terms to the Translation Manager. You can add new terms directly to the Translation Manager or by selecting terms from the Translation Terms Panel.

Let's add some terms directly to the Translation Manager.

1. From the menu bar at the top, go to **Tools > Translation Manager**.
2. The Translation Manager window will open, and you can view all the translatable terms. This example has terms that were already added and translated.
3. On the right side of the window, there is a list of icons where you can add, delete, import, export, and edit the Translation settings.

Translation Manager

Languages: (All) English Spanish

Translation Terms

Key	English (Alternate)	Spanish
+ Auto		Automatico
+ brake		freno
+ Brake Motors		Motores de Freno
+ Days		Dias
+ Hand		Manual
+ Humidity		Humedad
+ Last		Ultimo
+ Minutes		Minutos
+ Off		Apagado
+ Seconds		Segundos
+ Tank		Deposito
+ Temperature		Temperatura
+ Value		Valor

4. To add a new global term, click on the Add  icon. Enter the term (Tanks), and click **Save**.
5. Double click on the row of the new term, and under Spanish translation, enter the word "**Deposito.**" Click **Save**.

Translation Manager

Languages: (All) English Spanish

Translation Terms

Key	English (Alternate)	Spanish
+ Last		Ultimo
+ Minutes		Minutos
+ Off		Apagado
+ Seconds		Segundos
+ Tank		Deposito

Original Term
Tank

English (Alternate)

Spanish translation
Deposito

Save

6. To view all the global terms added to a project, go to **Tools > Translation Manager**. If you want to provide a translation to a term that has not been translated, double click on the row of the term, enter the translation, and click **Save**.

Translation Manager

Languages: (All) English Spanish

Translation Terms

Key	English (Alternate)	Spanish
+ Auto		Automatico
+ brake		freno
+ Brake Motors		Motores de Freno
+ Days		Dias
+ Hand		Manual
+ Humidity		Humedad
+ Last		Ultimo
+ Minutes		Minutos
+ Off		Apagado
+ Seconds		Segundos
+ Tank		Deposito
+ Temperature		Temperatura
+ Value		Valor

7. Now test it out. Drag a **Language Selector** component to your window. Go into **Preview Mode**, and toggle the languages. You will see the languages for the text switch between English and Spanish for these two components.



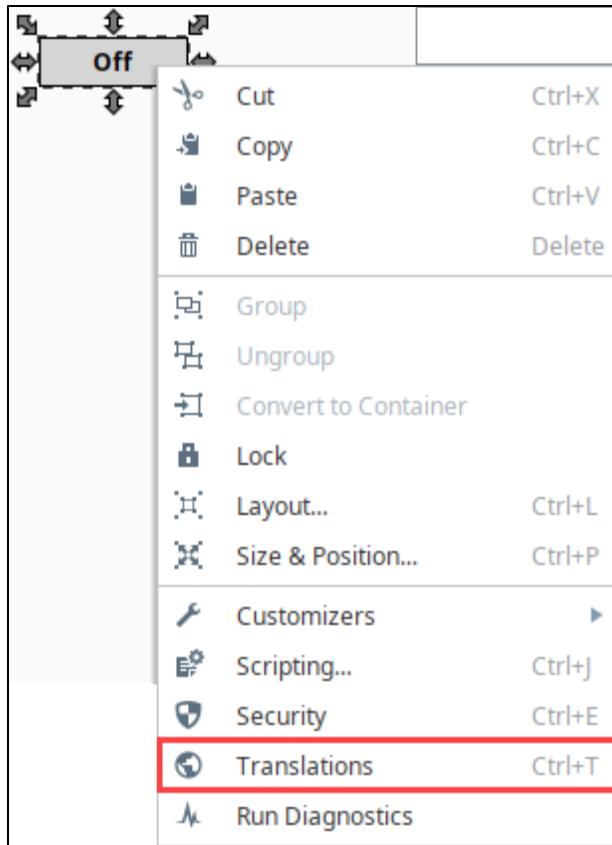
Add Component Level Translation Terms



Component level translation terms are only available in Vision.

What's nice about adding component translation terms is that the Translatable Terms panel provides all the terms for that component on the window. All you have to do is add the translations. The Translatable Terms Panel is a central place to manage all the translations associated with a specific component. One important reason to use a component level term is if it is going to be different from the global term. Just remember, that because a term is translated at the component level doesn't mean that other components get to use it. Typically, it is used for that component only.

1. In **Designer**, drag a **Multi-State Indicator** from the Component Palette in to your workspace.
2. Right click on the component and scroll down to **Translations**. The Translatable Terms panel will open and display all the terms for the specific component. Alternatively, you can right-click on the Root Container of a window and select **Translations**. The **Translatable Terms Panel** will display all the terms for all the components on the window.



Translatable Term Options

In the Translatable Terms Panel, you have the option of adding your translation to the Component, Translation Manager global database, or marking it for translation at a later time. You'll notice under the Component Terms section of the Translatable Terms window that there are four columns: Component Description, Key, Component translation, and Global translation.

You'll see below, all the terms for the Multi-State Button, and none of the terms have translations added. If you don't see the language you want your terms translated to, click the Earth icon , and select the language (i.e., Spanish). You will then see a column for your component translation and global translation.

Translatable Terms

The screenshot shows the 'Translatable Terms' panel with the title 'Component Terms'. It contains a table with four columns: Description, Key, Spanish (Co...), and Spanish (Gl...). The rows represent different text styles:

Description	Key	Spanish (Co...)	Spanish (Gl...)
Text	Off		Pending
Style State 0.Text	Off		Pending
Style State 1.Text	Auto		Pending
Style State 2.Text	Manual		
Style State 3.Text	Fault		
Style State 4.Text	Fault		
Style State 5.Text	Pending		

Selected Component: Multi-State Indicator

- To add translations for your component terms, double click the row of your term (i.e. ON). In the **Spanish Component** box, enter the translation (i.e. EN). You also have the option of creating a global translation on the Translatable Terms Panel as well. Then, click the **Back** link, and repeat this step for all your terms.

The screenshot shows two overlapping panels. The main panel is the 'Translatable Terms' panel with the title 'Component Terms'. A specific row for 'Style State 1.Text' is highlighted with a red box. The secondary panel, titled 'Translatable Terms' with a 'Back' link, shows the 'Original Term' as 'Auto' and the 'Spanish (Global)' field as empty. A red box highlights the 'Spanish (Component)' field, which contains the value 'Automatico'.

- Once you entered the translations for all your terms, you will see them on the Translatable Terms Panel in the Spanish (Component) column. If a term doesn't have a translation, you can select the term and mark it for translation by clicking the **Earth with a green plus icon**. This adds the term(s) to the translation database for someone else to add the translation.

In this example, you'll notice several things about each of the terms so look closely at them. You'll find examples of both component and global level terms, a component level term, a global term, a component term that overrides a global term, and terms marked for translation.

Translatable Terms

Component Terms Mark the selected term(s) for translation (adds them to the translation database)

Description	Key	Spanish (Com...)	Spanish (Glob...)
Text	Off		Pending
Style State 0.T...	Off		Pending
Style State 1.T...	Auto	Automatico	Pending
Style State 2.T...	Manual		
Style State 3.T...	Fault		Pending
Style State 4.T...	Fault		Pending
Style State 5.T...	Pending		

Selected Component: Multi-State Indicator

5. To test it out, drag a **Language Selector** component from the Component Palette to your workspace. Go to **Preview Mode**, select Spanish from the dropdown to see the text change from English to Spanish for these two components.



Translation List Import and Export

The ability to import and export terms and translations can be extremely important particularly to a large project. It allows you disseminate terms and translations rapidly, send them to a third party for translation, and keeps all related projects up-to-date.

Export Terms from a Window

Let's export some terms from the Translation Manager. In this example, you can see all the terms that were added to the Translation Manager. You can export selected terms, or all terms from the Translation Manager.

1. Go to Tools > Translation Manager. Select all terms or only specific terms to export. Click on the **Export Terms** icon on the right side of the **Translation Manager** window.

**INDUCTIVE
UNIVERSITY**

**Import and Export
Translations**

[Watch the Video](#)

Translation Manager

Languages	Translation Terms		
	Key	English (Alternate)	Spanish
<input checked="" type="checkbox"/> (All)	#HAND	Hand	Manual
<input checked="" type="checkbox"/> English	#MyText	<html>This is some long text and I...</html>	<html>Este es un texto largo yquier...
<input checked="" type="checkbox"/> Spanish	#OFF	Off	Apagado
	#ON	On	en
	#Tank1Name	Tank	Tanque 1
	#Tank2Name	Tank	Tanque 2
	#Auto		Automatico
	#Barrel	Barrel	Barril
	#Button		Tanques
	#Check Box		Rechazar
	#Confirm		
	#Controllers		Controladores
	#Goodbye		Adios
	#HAND		MANUAL
	#Hand		Manual
	#High		
	#Label		Tanque
	#Motors		Motores
	#OFF		APAGADO
	#Off	Off	Apagado
	#ON		EN
	#Sine0		Nivel
	#Sine1		Temperatura
	#Tab 1	Home	Casa
	#Tab 2	Office	Oficina
	#Tab 3	School	Escuela
	#Tank		Tanque
	#Tanks		Tanques
	#Temperature	Temperature	Temperaturas
	#Toggle Button	Motors	Motores

2. The Exported Terms window will open. Enter the following information:

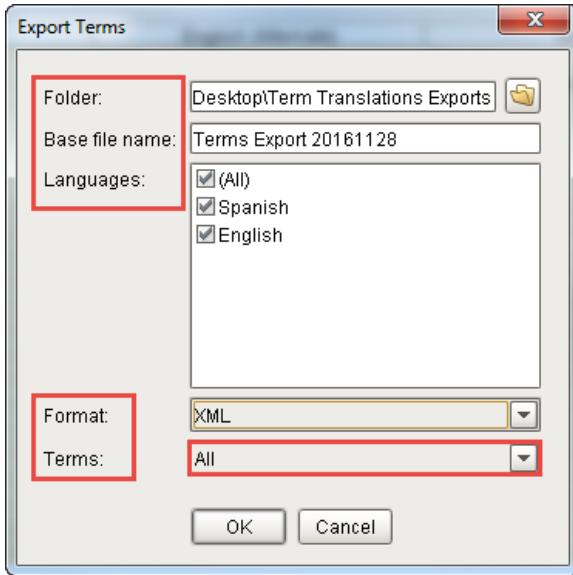
- a. Select the **Folder** location where you want to put your exported terms.
- b. Enter the **Base file name**. (File name of your exported terms).
- c. Select the appropriate **Languages**. You can have multiple languages selected depending on the number of language translations you are using (i.e., English, Spanish, and French). This example exports two files: one for the English translation, and one for the Spanish translation.
- d. From the **Format** dropdown list, select either **PROPERTIES** or **XML**. Both formats behave the same, the only difference is that the XML format supports UTF-8 encoding directly. A person can now translate the XML file directly and import it back into Ignition's Translation Manager.
- e. From the **Terms** dropdown list, choose either **Selected** or **All**.



For Specific Terms

If you want to select specific terms to export, choose '**Selected**', otherwise, '**All**' terms will be exported from the dropdown list at the bottom of the window.

- f. Click **OK**.



```

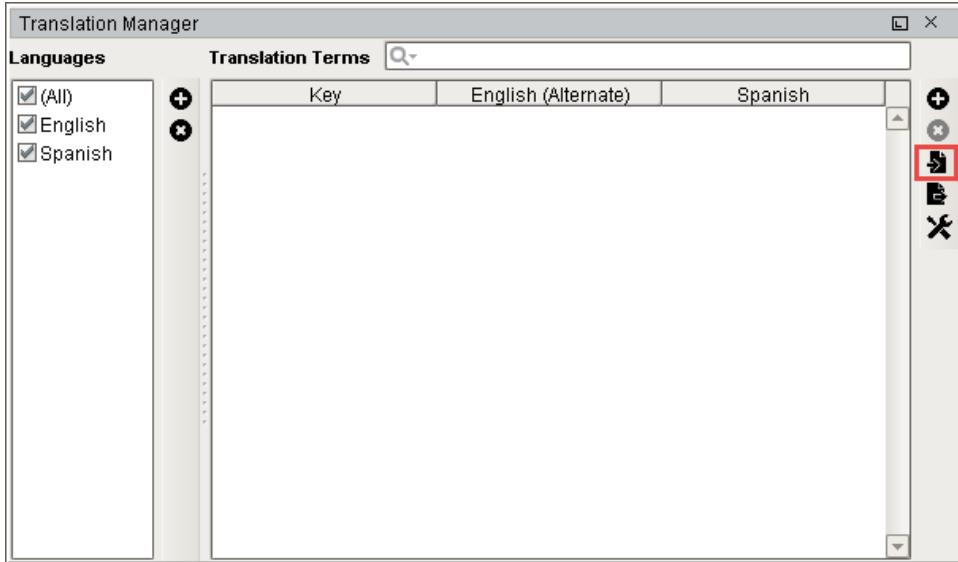
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
- <properties>
  <comment>Locale: en</comment>
  <entry key="#Sine1">Temperature</entry>
  <entry key="OFF"/>
  <entry key="Sine0"/>
  <entry key="Tanks"/>
  <entry key="#Tank1Name">Tank</entry>
  <entry key="Goodbye"/>
  <entry key="Toggle Button">Motors</entry>
  <entry key="Controllers"/>
  <entry key="#OFF>Off</entry>
  <entry key="Off>Off</entry>
  <entry key="Tab 3">School</entry>
  <entry key="#mytext"><html>This is some long text and I want it
    to wrap</entry>
  <entry key="Tab 2">Office</entry>
  <entry key="Tank"/>
  <entry key="Tab 1">Home</entry>
  <entry key="#HAND">Hand</entry>
  <entry key="Barrel">Barrel</entry>
  <entry key="Welcome"/>
  <entry key="Button"/>
  <entry key="Auto"/>
  <entry key="#ON">On</entry>
  <entry key="Hello"/>
  <entry key="Label"/>
  <entry key="High"/>
  <entry key="Hand"/>
  <entry key="Check Box"/>
  <entry key="#Tank2Name">Tank</entry>
  <entry key="HAND"/>
  <entry key="Motors"/>
  <entry key="Confirm"/>
</properties>

```

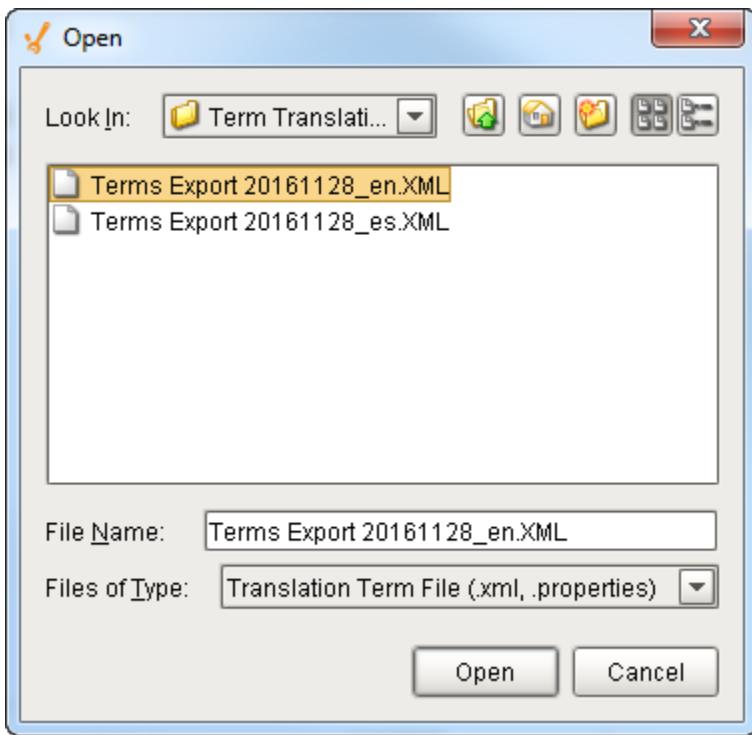
3. Go to the folder location and check to see if your exported files are there. You will see **one file exported file for each language**. Open the English exported file and verify that all the terms you intended to export are there.

Import Terms

1. Go to Tools > Translation Manager, and select the **Import Terms** icon from the right side of the Translation Manager window.



2. Navigate to the folder where your exported files reside, and choose the English file from the **Open** window, and click **Open**.

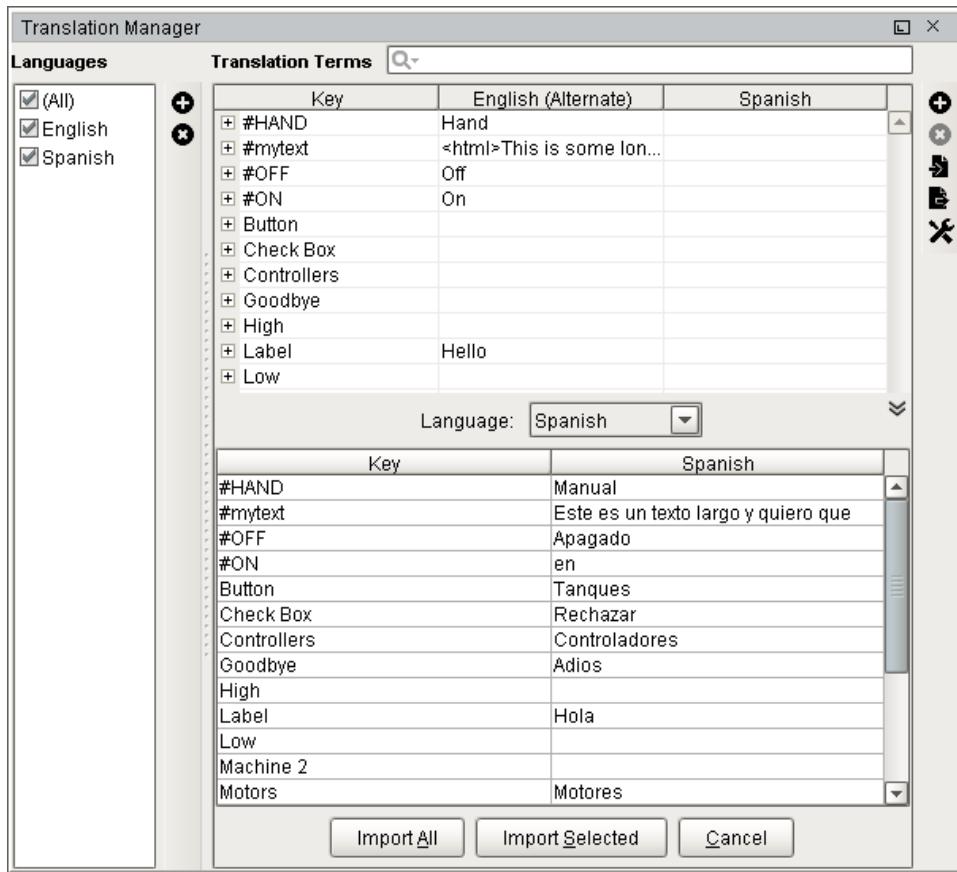


3. You have the option of importing selected terms or all terms. Click either **Import All** or **Import Selected** depending on what terms you want to import.

Repeat this step to import each translation file. This example imports the Spanish translation file.

Imported Terms Overwrite Existing Terms

Be aware that existing terms will be overwritten by the imported term.



4. Check to see that all your terms were imported into the Translation Manager by going to **Tools > Translation Manager**.

Related Topics ...

- [Switching the Current Language](#)
- [Localization Best Practices](#)

Switching the Current Language

Language Selection

Because Ignition has multiple visualization systems, each system has separate ways to switch the current language.

Vision

In Vision, the language can be selected using the language selector dropdown before logging in to the client, or can be added to the client in the form of the language selector component.

Perspective

In Perspective, the language is selected by changing the locale of the session by modifying the session.props.locale. This can be written to using a script when the session first opens, or something can be bound to it that the user can change.

Expression and Scripting Functions

You can look up translations using the following functions:

- Expression Function
translate()
- Scripting Functions
system.util.translate()
system.util.modifyTranslation() scripting functions.

Translations are matched by looking for the base language value in the translation database. This is especially useful for message boxes and other warnings or errors that you show in your scripts.

System Considerations During Translation

The inherent operating system may affect Ignition's ability to provide a complete translation in certain circumstances.

For example, suppose a user selected the Spanish option from their Language Selector component. An English to Spanish translation term exist for the word "Information" as well as "Start." The message box script takes two parameters: one for the content while the other parameter is optional. If the second parameter is not included in the function then the resulting title of the message box will include the English word "Information." The following code is executed on a button clicked event handler.

```
message = system.util.translate("Start")
system.gui.messageBox(message)
```

What results is a message box with a translated content while the title remains in English. In addition, the accept button is translated as well even though the English word for "OK" does not exist in the Translation manager. This is due to the system level translations that exist for inherent language support.

A similar occurrence exist with the file open dialog window. The following code is executed on a button clicked event handler.

```
system.file.openFile()
```

The result is a translation for the title as well as the open and cancel buttons while the references to the "File Name" and the "Files of Type" remain in the operating system's language and thus are beyond the scope of Ignition's ability to translate.

Similar occurrences will appear in print and error dialog boxes.

On this page ...

- [Language Selection](#)
 - [Vision](#)
 - [Perspective](#)
- [Expression and Scripting Functions](#)
- [System Considerations During Translation](#)

Localization Best Practices

Best Practices

Before you begin using multiple languages in your projects, we thought we would start you off with a few best practices for using project terms and translations.

As you already know, there are global level translations which are available to all components and text elements in both the Translatable Terms Panel and the global database, and component level translations that are only specific to the component and only available in the Translatable Terms Panel. It's good to understand a little about their behavior before investing a lot of manpower setting up your translations only to find out down the road that you might set them up differently after you become more familiar with them.

Global vs Component Terms

We recommend using global terms over component level terms because this way you will only have to add your term once since global terms are shared across all windows and projects. This will save you a little time from having to add the same term again if you discover that a component level term should extend beyond the specific component and shared across all projects. The only time you would want to use a component level term, is if you want it to be different from the global level term. This ensures that the component level term will override the global term.



Where the component level term takes precedence

The [component level term](#) always takes precedence when there is a matching global term.

Using Codes

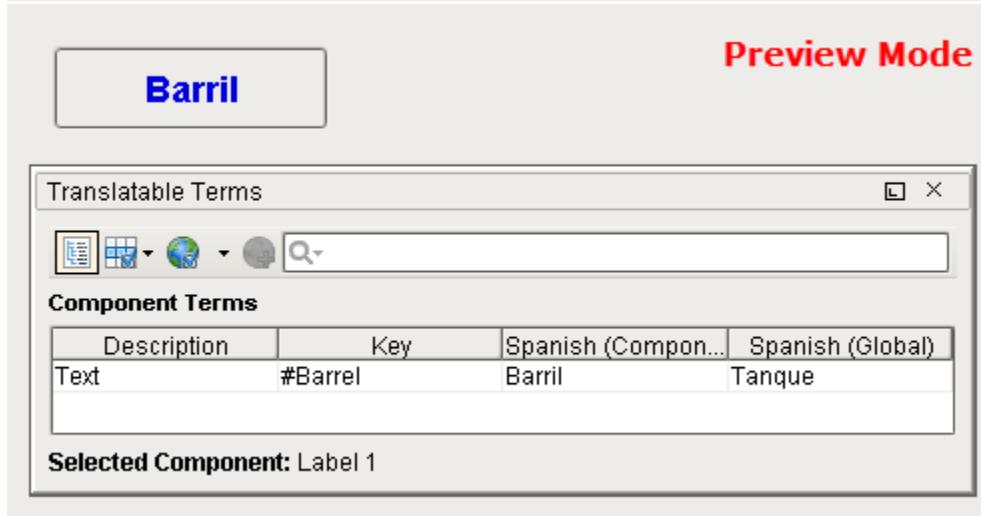
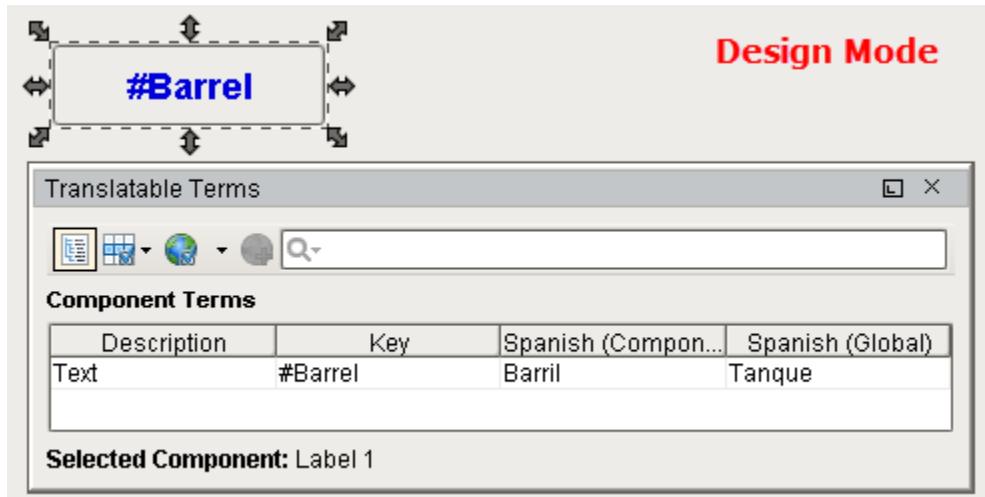
It's a good idea to use codes for the Key field of your Terms for any descriptions that you want to translate.(i.e: #introduction or #welcome_screen_info) so your global term won't ever accidentally translate on another window or component. (The '#' is just for convention, and is not a special character). Using codes works particularly well for long text strings such as introduction paragraphs. One thing to be very careful of is if you edit the codes in a minor way, it could potentially break the translation.

Here are a couple of examples:

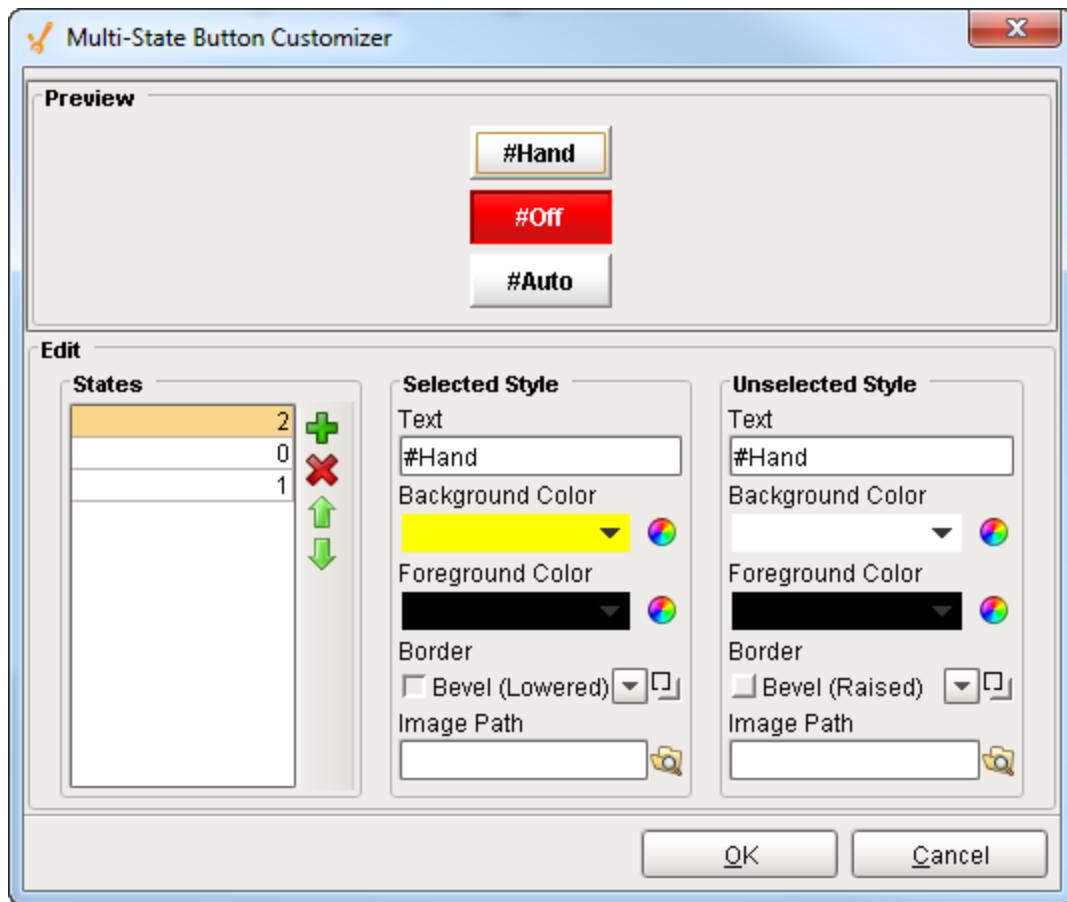
The first example is of a Label component with a Key field of "**#Barrel**." You'll notice the Label has global translation of "**Tanque**" and a component level translation of "**Barril**." By using the "#," the component level translation of "Barril" was not overwritten by the global translation of "Tanque" by using the "#." Here is what the translation looks like in **Design Mode** and **Preview Mode**.

On this page ...

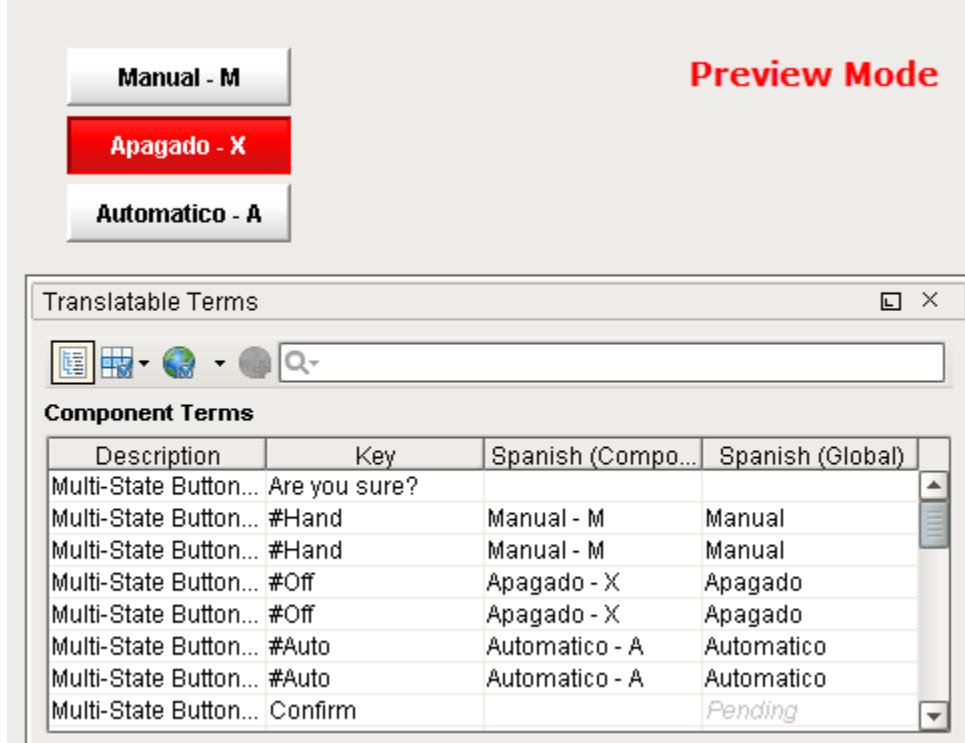
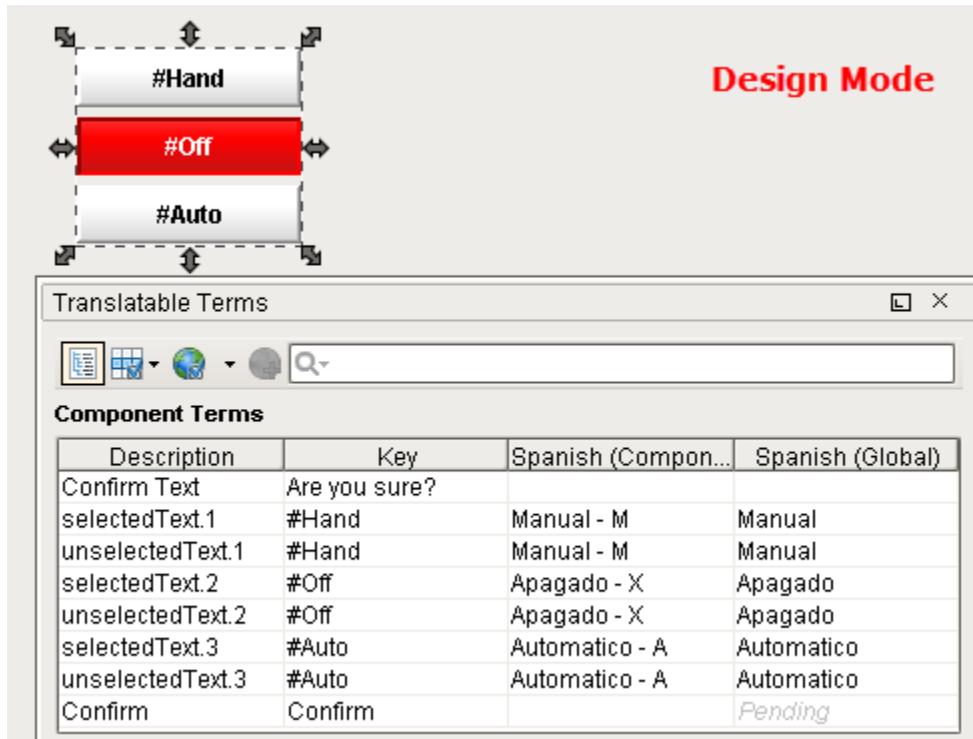
- [Best Practices](#)
 - [Global vs Component Terms](#)
 - [Using Codes](#)
 - [Using HTML for Text Wrapping](#)



The "#" can also be used on components for text that you don't want accidentally translated. This example uses the Multi-State Button component. The only way to change the states / words on the buttons (i.e., Hand, Off, Auto) is to use the Multi-State Button Customizer. Add the "#" to the text on each of the three Keys so they don't get translated by global terms that have already been set up.



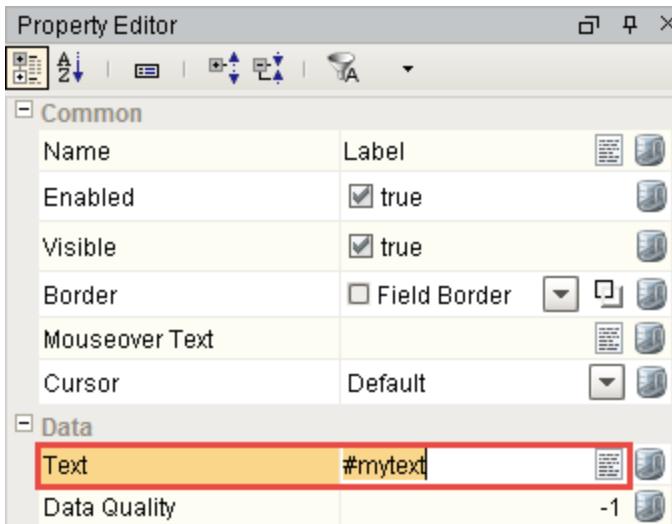
Here's what the translations look like in **Design Mode** and **Preview Mode**.



Using HTML for Text Wrapping

Another good practice is to use HTML to wrap text within your component's buttons, labels, etc.,. You can use it with translating terms that contain lots of text. HTML is just a good way to wrap text to fit within a label and button components.

1. In the **Property Editor** of a Label component, go to the **Text** property and give the Text a unique name (i.e., #Mytext).



2. Right click on the **Label** component, and click on **Translations**. Double click on the row for Mytext. Enter the text as shown below for both the **Spanish Global** and English **Global translations**. Make sure you precede your text with "<HTML>." Click the **Back** link.

```
Mytext

#English
<html>This is some long text and I want to wrap it

#Spanish
<html>Este es un texto largo y quiero que se envuelva
```

Design Mode

The screenshot shows the 'Translatable Terms' dialog in 'Design Mode'. At the top, there's a preview area with a dashed border containing the text '#mytext'. Below it is a table titled 'Component Terms' with one row:

Description	Key	Spanish (Comp...)	Spanish (Global)
Text	#mytext	<html>Este es un texto lar...	<html>Este es un texto largo y quiero que se envuelva

The 'Spanish (Global)' and 'English (Global)' fields both contain the same long HTML string: <html>Este es un texto largo y quiero que se envuelva. These two lines are highlighted with a red box.

3. Once in **Preview Mode**, your long text will be wrapped!



Translating Built-in Terms

There are several built-in terms used by Ignition that are translatable. This page contains the Keys that can be used to translate these terms.

Built-in Access Denied

When [Security Settings](#) are configured on a window, the only restriction is **Do Not Open**, which prevents the window from opening based on User Roles. The text in this message box is not normally configurable, but can be altered via translations using the following steps.

On this page ...

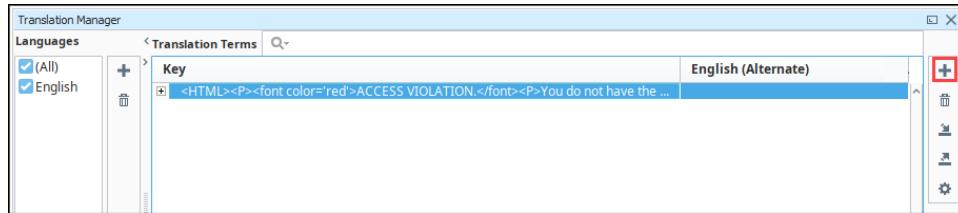
- [Built-in Access Denied](#)
- [Screen Locked Message](#)

1. Go to the top menubar in the Designer and select **Tools > Translation Manager**. This opens the Translation Manager window.
2. Copy the **Key** from the code block below, click the plus  icon and paste it in the **Add Item** field. Click **OK** and you'll notice the code will be displayed in the **Key** field.

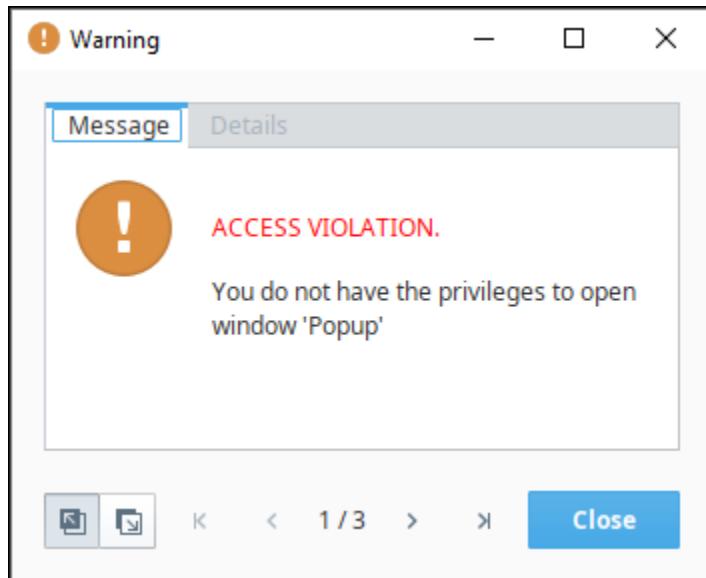
The line below should be used as the Key.

Access Violation - Key

```
<HTML><P><font color='red'>ACCESS VIOLATION.</font><P>You do not have the privileges to open window '%s'
```



3. Save your project.
4. When a user without the proper permission attempts to access the popup from the client, they will get the following message.



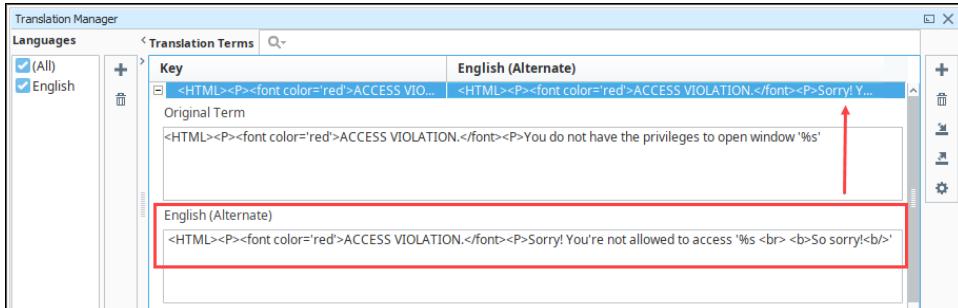
Using the same example from above, we modified a version of the text that appeared in the message box. We used the **English (Alternate)** as the language where the name of the window is passed into the message box via the '%s' substring.

1. Copy the new message from the code block below.

Access Violation - Translated Term

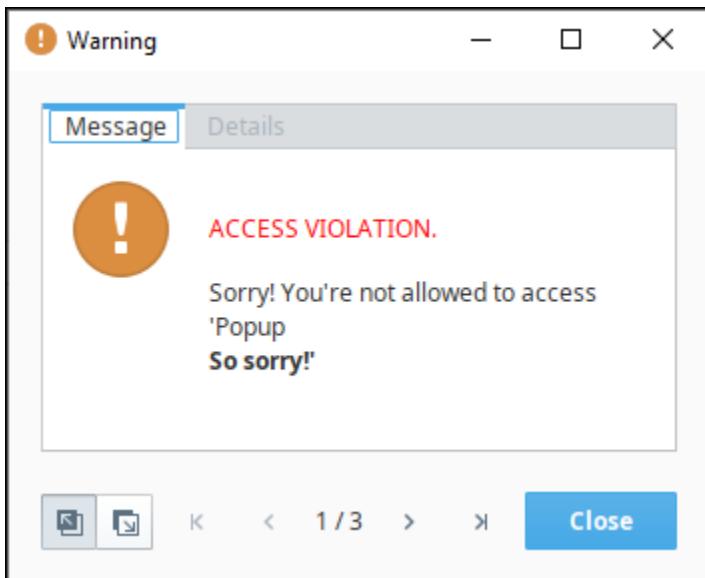
```
<HTML><P><font color='red'>ACCESS VIOLATION.</font><P>Sorry! You're not allowed to access '%s <br> <b>So sorry!<b/>'
```

2. In the Transaction Manager window, double click the row for the **Key** and paste it in the **English (Alternate)** field. Click **Save**.



3. Save your project.

Now, our message box appears as the following when a user does not have the proper access.



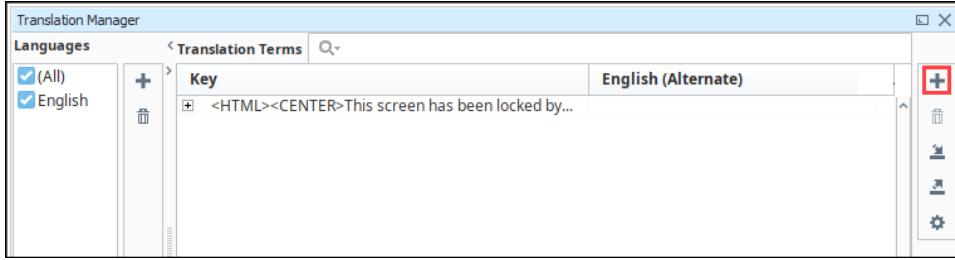
Screen Locked Message

The built-in [system.security.lockScreen](#) function in a [client timer script](#) can be used to lock the screen. In some cases, you may wish to change the text on the Screen Locked window. Use the following steps to change the text.

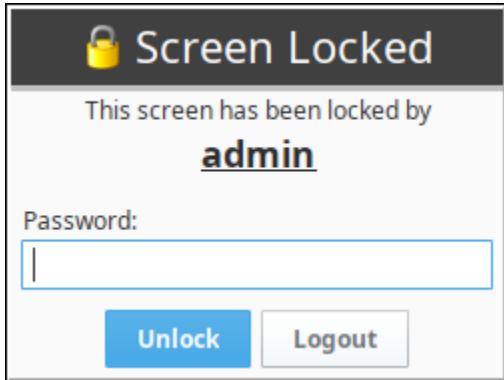
1. Open the Transaction Manger window.
2. Copy the **Key** from the code block below. Click the plus icon and paste it in the **Add Item** field. Click **OK** and you'll notice the code will be displayed in the **Key** field.

Screen Locked - Key

```
<HTML><CENTER>This screen has been locked by<BR><font size=+1><b><u>%s</u></b></font>
```



3. Save your project. Now the Screen Locked window will appear as follows.



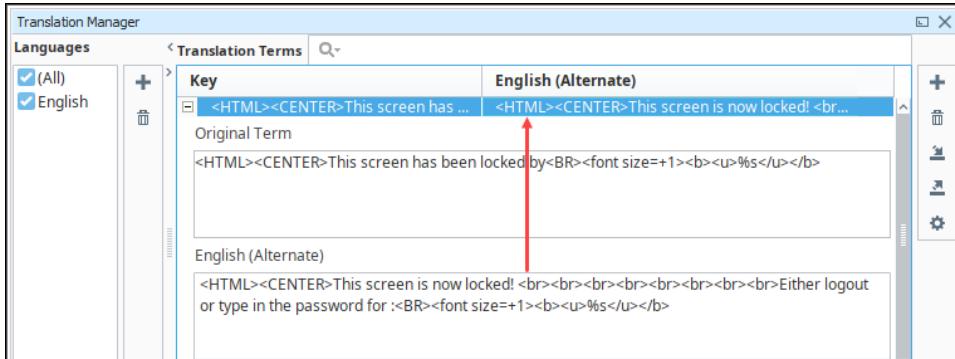
Building on the previous example, we can use the following key to translate the message on the Screen Locked window. The '%s' substring will reference the username of the user currently logged in. In this example, we added a few line breaks, which will make the Screen Locked window appear taller.

1. Copy the updated text from the code block below:

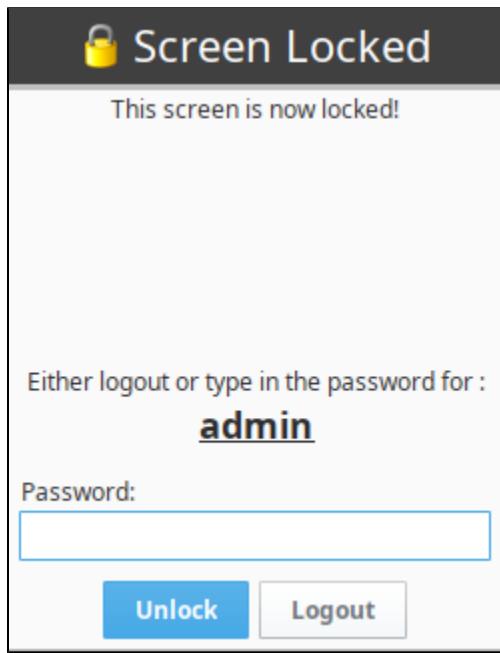
```
Screen Locked - Translated Term

<HTML><CENTER>This screen is now locked! <br><br><br><br><br><br>Either logout or type in the password for :<BR><font size=+1><b><u>%s</u></b></font>
```

2. In the Transaction Manager window, click the row for the **Key** and paste it in the **English (Alternate)** field. Click **Save**.



3. Save your project. Now, the message on the Screen Locked window appears as follows.



Expression Language and Syntax

Expression Language

The expression language is used to define dynamic values for component properties and expression Tag s. Expressions often involve one or more other values that are used to calculate a final value. In most cases, expressions only return a value.

The classic example for an expression is to change a temperature that is stored in Celsius to Fahrenheit in order to display it. Suppose you had a Tag, Tank 6/Temp, that was in Celsius. If you wanted to display that Tag in Fahrenheit on a Label, you would use an expression binding on the label's text property using the following expression:

```
1.8 * {Tank 6/Temp} + 32
```

Every time that the temperature Tag changes, the expression will re-calculate the value and push it into the Label's text property. Now lets say that you wanted to append a "°F" to the end of the label so that the user knew the units of the temperature. You could simply use some string concatenation in your expression, like this:

```
(1.8 * {Tank 6/Temp} + 32) + " °F"
```

Lets suppose that you wanted to give the user an option to display the value in Celsius or Fahrenheit, based on checking a checkbox. You could add a [Check Box](#) component to the screen called DisplayFahrenheit. Then you could use this expression to dynamically display either unit, based upon the user's selection:

```
if({Root Container.DisplayFahrenheit.selected}, (1.8 * {Tank 6/Temp} + 32)  
+ " °F", {Tankf/Temp} + " °C")
```

For more information see [Expression Bindings in Vision](#) or [Expression Bindings in Perspective](#).

Syntax

As its name suggests, everything in the expression language is an "expression". This means that everything returns a value: 5 is an expression, so is 5 +1, and so are {MyTags/TankLevel} and {MyTags/TankLevel}+1. Expressions can be combined in many powerful ways. Lets take a look at how expressions are written.

More formally, an expression is any one of the following:

- Number
- Boolean
- String
- Bound Tag
- Bound property
- Function call
- Dataset access
- Equation involving any of these

Literal Values

Literal values are things like numbers, booleans, and strings that are represented directly in the language. In the expression language, numbers can by typed in directly as integers, floating point values, or using hexadecimal notation with a 0x prefix. Examples:

```
42  
8.456  
0xFFC2
```

Strings are represented by surrounding them with double or single quotes. You can use the backslash character to escape quotes that you want to be included in the string. Examples:

On this page ...

- Expression Language
- Syntax
 - Literal Values
 - Operators
 - Bound Values
 - Dataset Access
 - Expression Functions
 - Whitespace and Comments
 - Tag Paths
- Additional Examples
 - String Concatenation
 - Celsius to Fahrenheit
 - Format Date
 - Date Manipulations
 - Bit Functions
 - Switch
 - Checking Conditions

```
"This is a regular string"
'This one uses single quotes'
"This string uses \"escaping\" to include quotes inside the string"
```

In addition, the following escape characters are available:

Character	Description
\n	New line
\r	Tab
\t	Carriage return

```
// The words "Hello" and "User" will be placed on separate lines
"Hello\nUser"

// Each "\t" will inject a tab
"Lots\tof\tSpace"
```

Operators

You can use these arithmetic, logical, and bit-shifting operators to combine expressions.

Operator	Name	Description
//	Comments	Allows for comments following this operator.
-	Unary Minus or Subtraction	If both preceded by a number, then returns a value by subtracting the operand right from the operand on the left of the operator: If preceded by anything else (or nothing, such as the start of the expression) and is followed by a number, this operator will negate the number. <pre>// This will return 6. 10 - 4 // This will return -10. - (15 - 5)</pre>
!	Not	Logical opposite of a boolean.
^	Power	Raises a number to the power of another number.
%	Modulus	Modulus or remainder of two numbers. a%b is the remainder of a÷b.
*	Multiply	Multiplies the number on the left of the operator by the number on the right of the operator.
/	Divide	Divides the number on the left of the operator by the number on the right of the operator.
+	Add or Concatenation	If both operands are numbers, this will add them together. Otherwise treats arguments as strings and performs concatenation. <pre>// This will return 10. 4 + 6 // This will return 'FirstSecond'. 'First' + 'Second' // This will return '2Alarms'. 2 + 'Alarms'</pre>
-	Subtraction	Subtracts the number on the right of the operator from the number on the left of the operator.

<code>~</code>	Bitwise NOT	Examines the bits of an operand, and performs logical negation: bits with a value of 0 become 1, and vice versa.
<code>&</code>	Bitwise AND	<p>Examines the bits of two operands, and performs a logical AND to each set, comparing the bits in each position from both sets. Returns true for any position where the bits being compared are true.</p> <pre>// 0101 // AND 0011 // = 0001 // Performs the bitwise operation above, resulting in the decimal 1. 5 & 3</pre>
<code> </code>	Bitwise OR	<p>Examines the bits of two operands, and performs a logical OR to each set, comparing the bits in each position from both sets. Returns true for any position where either bit being compared is true.</p> <pre>// 0101 // OR 0011 // = 0111 // Performs the bitwise operation above, resulting in the decimal 7. 5 3</pre>
<code>xor</code>	Bitwise XOR	<p>Examines the bits of two operands, and performs a logical exclusive OR to each set, comparing the bits in each position from both sets. Returns true for any position where only one of the bits are true.</p> <pre>// 0101 // XOR 0011 // = 0110 // Performs the bitwise operation above, resulting in the decimal 6. 5 xor 3</pre>
<code><<</code>	Left Shift	A signed bitwise left shift.
<code>>></code>	Right Shift	A signed bitwise right shift.
<code>></code>	Greater Than	Logical greater-than test between two numbers. Returns a boolean.
<code><</code>	Less Than	Logical less-than test between two numbers. Returns a boolean.
<code>>=</code>	Greater Than or Equal To	Tests if the operand on the left is greater or equal to the operand on the right. Returns a boolean.
<code><=</code>	Less Than or Equal To	Tests if the operand on the left is less than or equal to the operand on the right. Returns a boolean.
<code>=</code>	Equal	Tests for equality between two operands.
<code>!=</code>	Not Equal	Tests for equality, returning true when not equal.
<code>&&</code>	Logical AND	Returns true when both operands are true. Anything non-zero is considered true.
<code> </code>	Logical OR	Returns true when either operand is true. Anything non-zero is considered true.
<code>like</code>	Fuzzy String Matching	Compares the left-hand value with the pattern on the right side. The pattern may consist of %, *, and ? wildcards.

Bound Values

Bound values are paths to other values enclosed in braces. These will appear red in the expression editor. When you are writing an expression for a [Expression Binding in Vision](#), you can reference Tag values and property values using the brace notation. When you are writing an expression for an expression Tag, you can only reference other Tag values. You can use the **Insert Property**  icon and **Insert Tag**  icon to build these references for you.

Dataset Access

If you have an expression that returns a dataset, you can pull a value out of the dataset using the dataset access notation, which takes one of these forms:

```
Dataset_Expression ["Column_Name"]      //Returns the value from the first row at the given column name.  
Dataset_Expression [Column_Index] //Returns the value from the given column at the first row.  
Dataset_Expression [Row_Index, "Column_Name"] //Returns the value from the given row at the given column name.  
Dataset_Expression [Row_Index, Column_Index] //Returns the value from the given row at the given column index.
```

For example, this expression would pull a value out of a [Table](#) at row 6 for column "ProductCode":

```
{Root Container.Table.data}[6, "ProductCode"]
```

Note that you'll often have to convince the expression system that what you're doing is safe. The expression language can't tell what the datatype will be for a given column, so you may have to use a type-casting function to convince the expression language to accept your expression, like this:

```
toInt({Root Container.Table.data}[6, "ProductCode"])
```

Expression Functions

The expression language's functions are where much of the real power lies. A function may take various arguments, all of which can themselves be any arbitrary expression. This means that you can use the results of one function as the argument to another function. In general, the syntax for a function call is:

```
functionName(expression1, expression2, ...)
```

Whitespace and Comments

Whitespace, such as spaces, tabs and newlines, are largely ignored in the expression language. It is often helpful to break your expression up onto multiple lines for clarity. Comments are delimited by two forward slashes. This will make the rest of that line be ignored. This example shows an if function spread over four lines with comments annotating the arguments.

```
if( {Root Container.UseTagValueOption.selected},  
    {MyTags/SomeValue}, // Use the Tag value.  
    "Not Selected"    // Use default value if the user doesn't check the box.  
)
```

Tag Paths

While referencing a Tag path in expressions, you may see some special notation, such as "~" and "[.]". More information on this notation can be found on the [Tag Paths](#) page.

Additional Examples

The following headings demonstrate many simple examples that better demonstrate the expression language's syntax.

String Concatenation

You can use an expression binding to concatenate strings resulting in a new string that reflects the concatenation of different strings.

The function starts with `CONCAT()` and inside the function is a list of strings. They can be manually typed like "42" or they can come from Tags or properties.



Expression Binding – Concat Strings

[Watch the Video](#)

Example 1

```
concat("The answer is: ", "42") //returns "The answer is: 42"
```

Example 2

You have a date, and need to extract the year, and concatenate the word "Vintage" to the end for a label display. Bind a label's text property to:

```
dateExtract({Root Container.VintageDate}, 'year') + ' Vintage'
```

Celsius to Fahrenheit

Use an expression binding to convert a temperature from Celsius to Fahrenheit. This is an example of how an expression binding can handle calculations.

```
{celsiusTemp} * 9/5 + 32
```

The reference to the Celsius temperature can come from a property or a Tag. As the property or the Tag changes, so does the expression binding.



Expression Binding - Celsius to Farenheit

[Watch the Video](#)

Format Date

You can format a date in an expression binding by using the `dateFormat` and `now` functions.

To make a label that updates to show the current time:

1. Drag a **Label** component onto the window.
2. Select the label's **Text** property binding icon and select **Expression** binding.
3. Enter the following code into the expression, and click **OK**.

```
dateFormat(now(1000), "MMM d, yyyy hh:mm:s a")
```

The `dateFormat` function takes two arguments. The first argument is any date type variable. This can include another function (like `now`) that returns a date type. The second argument refers to the date format that you want returned. The `now` function returns the current time and in this case it will update every second.

For more information on the date formatting expression, see the [Appendix](#).



Expression Binding – Format Date

[Watch the Video](#)

Date Manipulations

You can manipulate dates in expression bindings such as a date addition/subtraction with the `dateArithmetic` function. This is important when you want to use the expression bindings to select a date that is offset by a certain amount.

The following example will return the time 15 minutes ago by using the `dateArithmetic` expression function:

```
dateArithmetic(now(), -15, "minute")
```



Expression Binding – Date Manipulations

[Watch the Video](#)

Bit Functions

You can use various bit functions in expression bindings like `getBit` to return individual bits of a word.

Example 1

Assuming a Tag path 'Folder/BitTag', the following would return the binary representation of the Tag's value at the 0 position

```
getBit({Folder/BitTag}, 0)
```

For more details on the `getBit` function, please see the [getBit page](#) in the Appendix.

Example 2

You have 3 bits in a PLC, only one of which will be on at a time. You want to turn these 3 bits into a single integer (0,1,2) to drive a component's Styles. Bind a custom integer property to:

```
binEnum({MyTags/Bit1}, {MyTags/Bit2}, {MyTags/Bit3})
```

Switch

You can use the switch function in expression bindings to evaluate conditional statements. This function acts like the switch statement in C-like programming languages. It takes the value argument and compares it to each of the case1 through caseN expressions.

The following example returns the string "Running" when it is given the value of 1. Its options are 0,1, and 2. And when comparing the value to the options the switch statement returns one of the corresponding results. If a result cannot be found, a fail-over option is returned.

```
switch(
1,
0, 1, 2,
"Off", "Running", "Fault",
forceQuality("!BAD STATE!", 0))
```

Checking Conditions

You can use expression bindings to return true or false based on different conditions.

Example 1

Consider the following expression that references the Tag with a path of 'Folder/Machine State':

```
{Folder/Machine State} = 0
```

The above expression simply tests the value of the Machine State Tag. If the value of the Machine State Tag is ever equal to 0 then the above expression would return true. In every case where Machine State is not equal to 0, then the expression would return false.

Example 2

It is possible to check for multiple conditions in the same expression. If you have two boolean Tags and you only want the expression to return true if both Tags are true then the binding would look like:

```
{boolTag1}=True && {boolTag2}=True
```

Example 3



INDUCTIVE
UNIVERSITY

Expression Binding - Bit Functions

[Watch the Video](#)



INDUCTIVE
UNIVERSITY

Expression Binding – Switch

[Watch the Video](#)



INDUCTIVE
UNIVERSITY

Expression Binding - Checking Conditions

[Watch the Video](#)

Consider the following expression that references the Tag with a path of 'Folder/Machine State':

```
if({Folder/Machine State} = 0, 1, 0)
```

This also finds the opposite of the Machine State if it's a boolean. It returns whether or not the Machine State is 0.

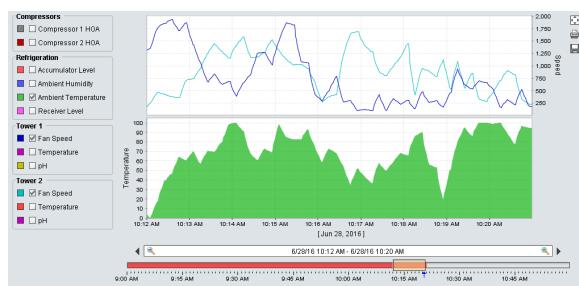
SQL in Ignition

Leveraging Databases

In addition to all the normal HMI functionality you would expect, Ignition has the ability to connect to databases, and this can greatly increase the functionality available to you! You can use databases to store history, create easy to search lists and configurations, and retrieve data from ERP or other systems. When you start using SQL, you can expand your project from a simple HMI to a project that brings your whole process together. The best part is that Ignition connects to as many databases as you want, all from one central location. Because the database lives outside of Ignition, we don't maintain any control or rules over what you can do with your data.

Displaying Data

You can easily display information from your databases on a window along with anything else in Ignition. You can show parts lists, step sequences, [realtime or historical charts](#), add the ability to search for inventory, or anything else you can think of.



What is SQL?

SQL stands for "Structured Query Language" and is the backbone of most modern relational databases. It's often referred to as "S.Q.L." or "Sequel," but both are correct and widely recognized. This language (different than the [Python Scripting Language](#)) allows you to write out requests or "queries" against the existing data to [view, add, edit, or remove](#) the information you want in a simple format.

Everything in a relational database is based around tables. Tables store the basic information for any system and can be combined together to make very efficient queries to retrieve your data.

	id	name	description	location	qty
▶	576	Mold 1	Mold for part # 1523	Warehouse A	23
	577	Mold 2	Mold for part # 8825	Warehouse A	12
	578	Bin 1	Finished Goods Bin 1	Warehouse B	9
	579	Bin 2	Scrap Bin 1	Warehouse B	14

```
SELECT *
FROM inventory
JOIN location ON location.name = inventory.location;
```

SQL Queries

SQL queries are crucial to Ignition's database-centric model. Queries can [show what is available](#) or [alter data in the databases](#), and some companies have positions just dedicated to running databases and creating queries. Anywhere Ignition is fetching data, you can choose to use your own custom queries to get exactly what you want out of the database. You can make your queries as simple or complex as you like. If your database is large, you might have a whole team dedicated to creating these queries for you and Ignition will happily execute them.

On this page ...

- [Leveraging Databases](#)
 - [Displaying Data](#)
 - [What is SQL?](#)
 - [SQL Queries](#)
 - [Database Connections](#)
 - [Using SQL in Ignition](#)
 - [Queries in Bindings](#)
 - [Queries in Scripting](#)
 - [Queries in Tags](#)
 - [Queries in Reports](#)
 - [Queries in Transaction Groups](#)
 - [Database Query Browser](#)
 - [Auto Generated Queries](#)
 - [Other Resources for Learning SQL](#)

SQL - Select Statement - Select Data from a Table

```
SELECT * FROM mytable
```

SQL - Inner Join Statement - Selects Records that have Matching Values

```
SELECT users.id, users.firstname, users.lastname, roles.name as 'rolename'  
FROM users  
INNER JOIN mapping ON users.id = mapping.userid  
INNER JOIN roles ON mapping.roleid = roles.id  
WHERE roles.name = 'Administrator'
```

Database Connections

Any SQL query you use needs a [Database connection](#), but Ignition simplifies all that by creating database connections in the Gateway instead of in the clients. This means from one central location you can manage all your database connections, and you don't have to worry about planning around adding clients in the future. Any special rules or connection restrictions are taken care of in the Gateway.

Using SQL in Ignition

There are many types of queries, and many ways to use them in Ignition. Some provide an easy to use builder to automatically store or fetch data, and some allow you to completely customize your queries.

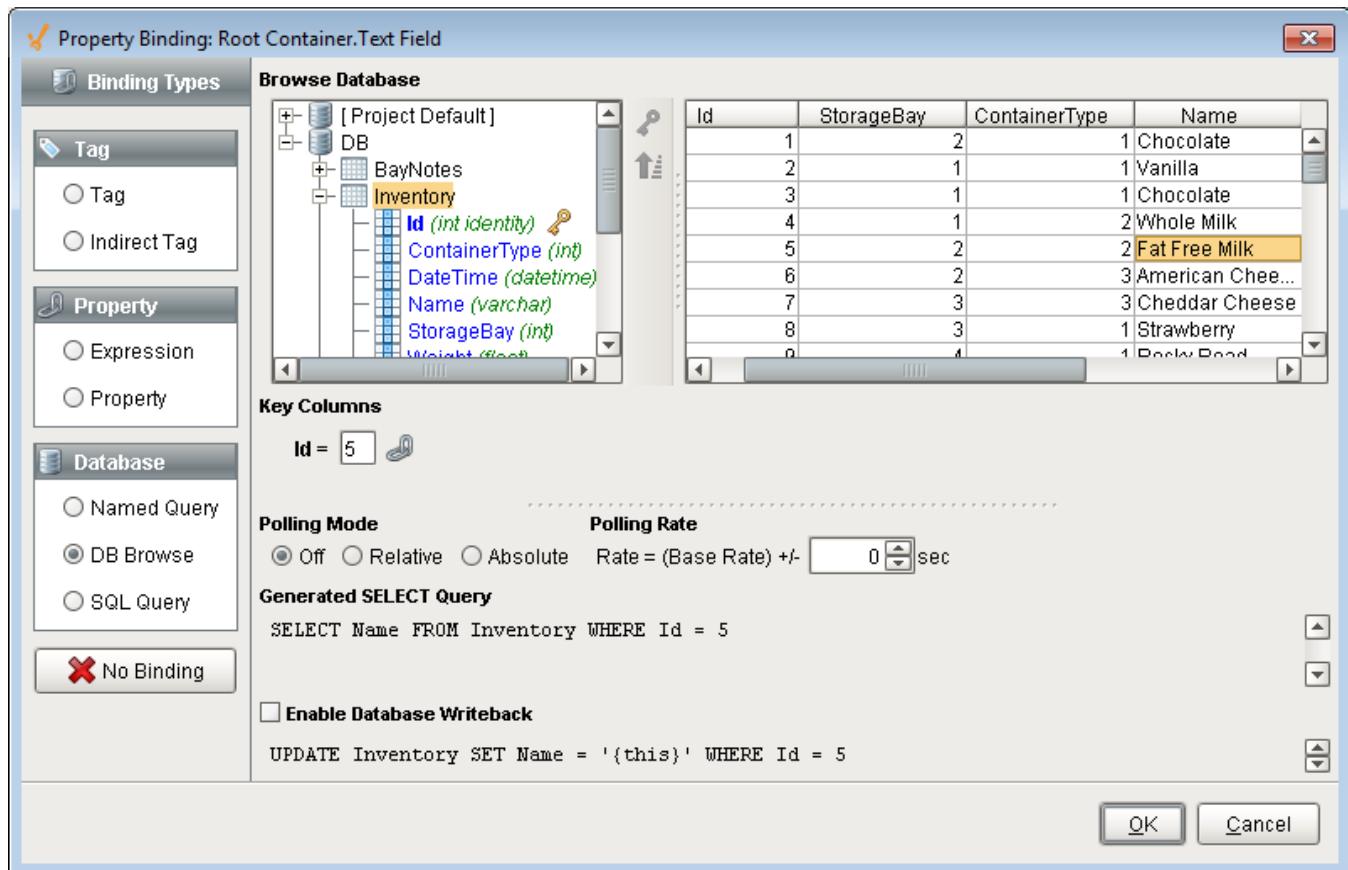
Queries in Bindings

Ignition's binding system offers a lot of flexibility in how database data can be used in a binding. The [Named Query](#) binding allows you to select one of the Named Queries that were previously built for that project, offering a very secure method of pulling data from the database. The [DB Browse](#) builder provides an interface that will build the query based on the data in the table that was selected. This allows even users with little SQL knowledge to pull data from the database. Finally, the [SQL Query](#) option will accept a straight query, so that a query specific to that binding can be written directly in the binding. When binding to a basic (non-dataset) data type, you can use the Writeback feature directly to send any changes back to the database.



Querying Data from Database

[Watch the Video](#)



Queries in Scripting

Ignition offers a number of built-in scripting functions for you to use to query your databases. This makes it very simple to view data, create dynamic scripts that use real data, and more. You can pull individual pieces of information, return whole tables of data, or update your database directly. Depending on the type of query and the sort of results you want, you will use different functions. The following functions are the ones you will use most, and all of them can use a special placeholder (?) to allow for dynamic query building.

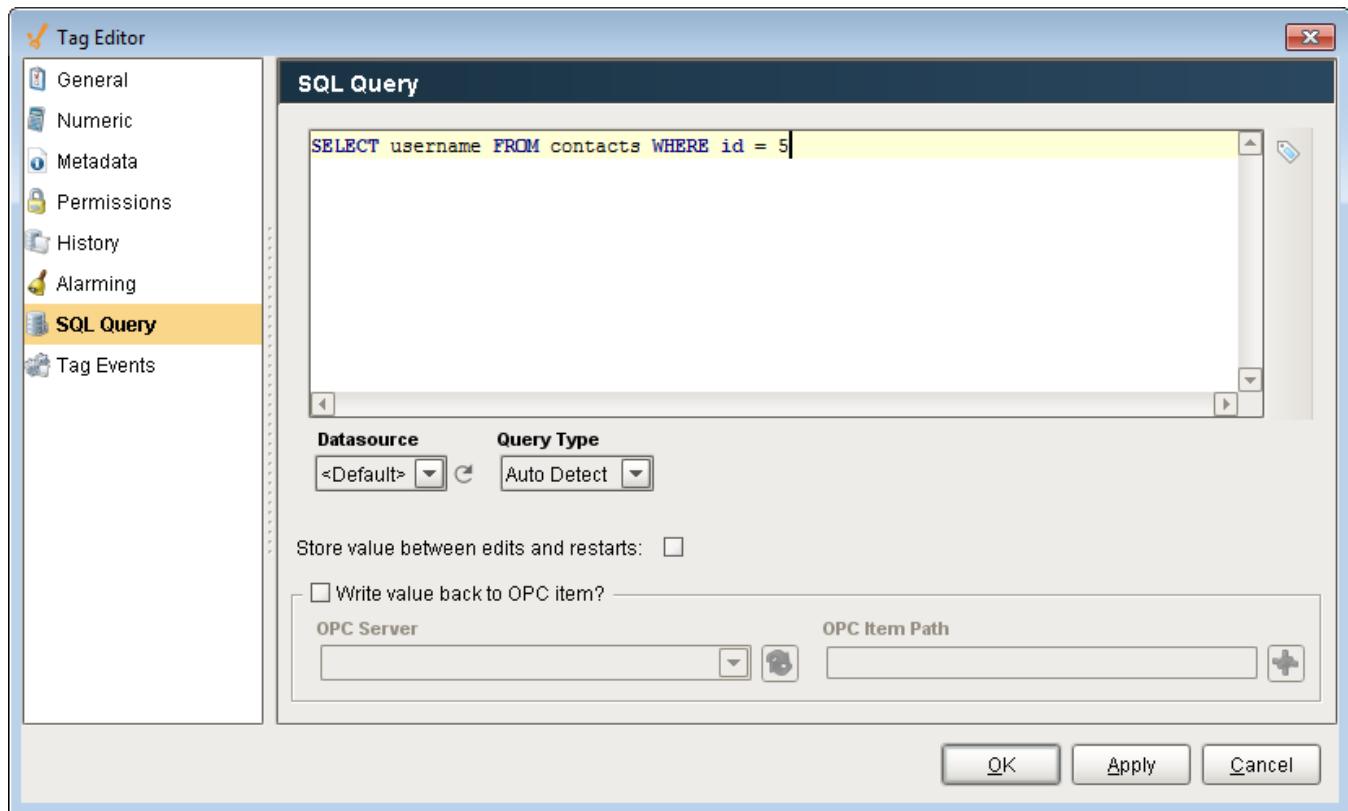
Scripting Function	Description
<code>system.db.runNamedQuery()</code>	Used to run a previously setup Named Query from within a script.
<code>system.db.runPrepQuery()</code>	Used to run basic SELECT queries to fetch whole datasets. This can be used to populate tables, or to sift through the data to do your own calculations.
<code>system.db.runPrepUpdate()</code>	Used to run queries that change the data in the database. Usually used on input form windows to update your database records.
<code>system.db.runScalarPrepQuery()</code>	Used when you want only one value from your results. Perfect for fetching a single value like the highest ID, or first timestamp of a result set.

Each of the different functions takes in different arguments (values) and provides slightly different options and functionality. For example, the `runPrepUpdate()` can return the auto-generated key from insert queries. This can be extremely helpful and eliminate the need to hit the database multiple times if you are using linked tables.

You can find examples of each of these and all the other database functions in the [system.db](#) section of the appendix.

Queries in Tags

Ignition offers Query Tags, which can run queries and return the result as a Tag value, giving all of the projects in the Gateway access to the same Database values.



Queries in Reports

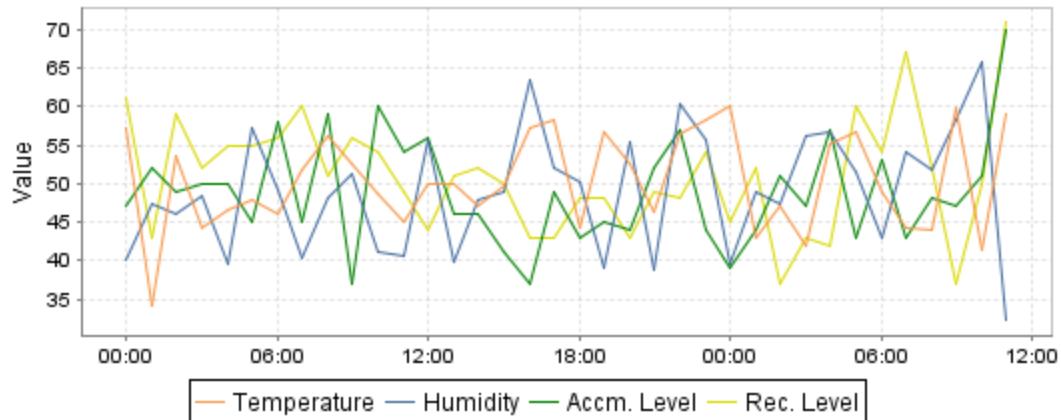
You can leverage queries to access data from all database connections to [create rich reports](#), from simple data logging to complex reports with grouped charts and datasets.

Tag History Report

A report that queries tag history and displays a graph and table

Report Start: 6/27/2016 12:00 AM

Report End: 6/28/2016 11:00 AM



Date / Time	Temperature	Humidity	Accm. Level	Rec. Level
6/27/2016 12:00:00 AM	57.34	40.14	47	61
6/27/2016 1:00:00 AM	33.98	47.33	52	43
6/27/2016 2:00:00 AM	53.61	46.15	49	59
6/27/2016 3:00:00 AM	44.3	48.5	50	52
6/27/2016 4:00:00 AM	46.57	39.59	50	55
6/27/2016 5:00:00 AM	47.9	57.16	45	55
6/27/2016 6:00:00 AM	46.04	49.16	58	56
6/27/2016 7:00:00 AM	51.87	40.42	45	60
6/27/2016 8:00:00 AM	56.24	48.07	59	51
6/27/2016 9:00:00 AM	52.42	51.17	37	56
6/27/2016 10:00:00 AM	48.74	41.01	60	54
6/27/2016 11:00:00 AM	44.89	40.67	54	49
6/27/2016 12:00:00 PM	49.85	55.73	56	44
6/27/2016 1:00:00 PM	49.82	50.94	46	51

Queries in Transaction Groups

While [Transaction Groups](#) are great at storing Tag data to a Database automatically, the built-in [Expression Items](#) can execute a SQL Query within the Transaction Group.

Group

Execution Disabled **Save project to apply changes** Enabled **Disabled** Pause

Edit group tag

Expression/SQL

Expression Type *: SQL Query

```

SELECT
    COUNT(target_number)
FROM
    workorders
WHERE
    id = { [~]currentWoID}
  
```

Datasource: <Default> **Query Type**: Auto Detect

OK **Apply** **Cancel**

Trigger **Action**

Execution Scheduling:

- Timer Schedule
- 1 second(s)

Update mode: OPC to DB

Data source: <Default>

Table name: group_table

Automatically create

Use custom index co

Store timestamp to:

Store quality code to

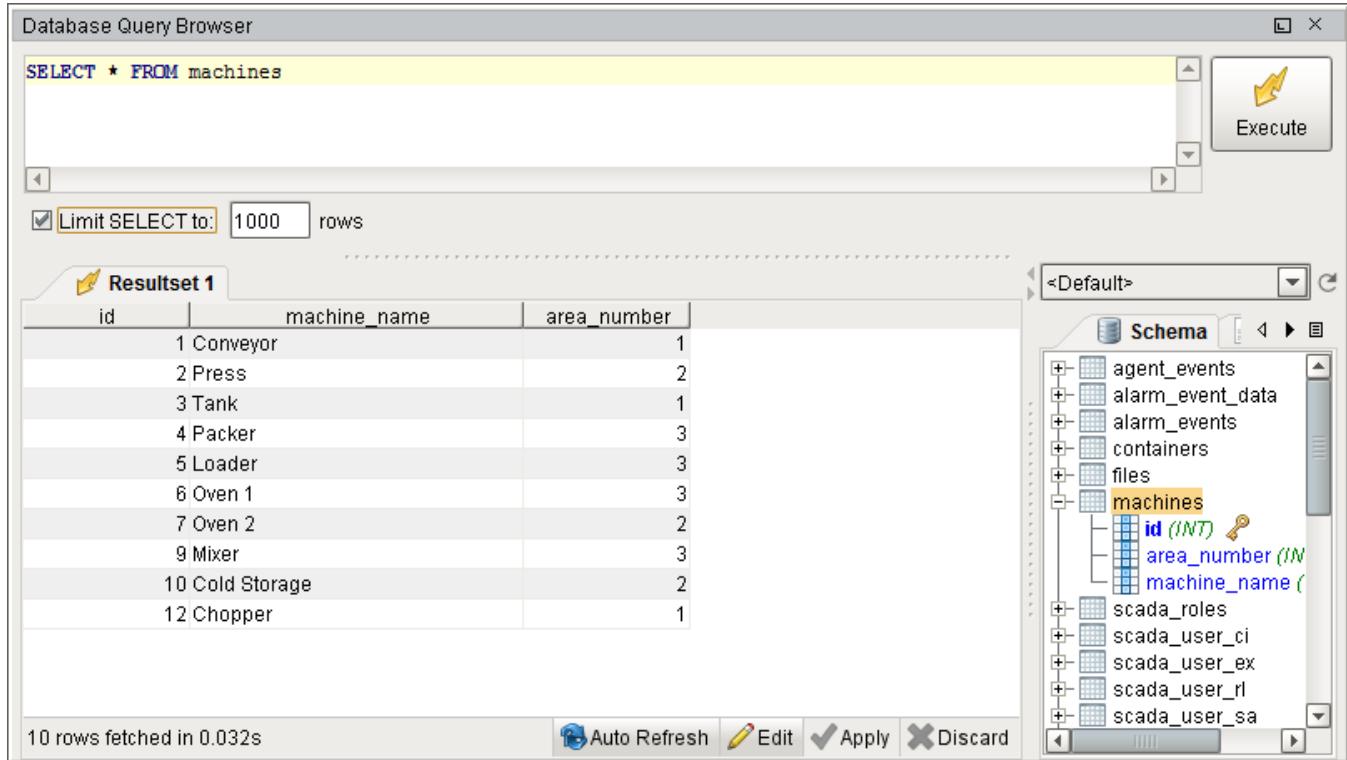
Delete records older

1 day(s)

Table action

Database Query Browser

The [Database Query Browser](#) offers an easy to use environment to run queries in for testing. Here, queries can be tested to figure out what values get returned, or data can be updated through raw queries, or the Database Query Browsers easy to use GUI editor.



Auto Generated Queries

Many systems within Ignition utilize a database connection, but the queries that are executed are constructed automatically by the system and do not require you to build the queries manually. These systems such as the [Tag Historian](#), the [Alarm Journal](#), or the [Database User Source](#) are very easy to setup and use since each system will automatically generate the necessary tables in the database, insert the relevant data, and even has prebuilt tools to extract the data. However, it is important to note that while these systems can automatically generate queries for you using the various components, these systems are simply storing data in a database which you can manually query out by building your own SQL queries.

Other Resources for Learning SQL

The following resources are unrelated to Inductive Automation and in no way affiliated, but provide good tutorials and basic SQL querying and syntax.

- [W3Schools SQL Tutorial](#)
- [SQL commands and syntax](#)
- [SQLZOO Tutorial](#)

Related Topics ...

- [Database Connections](#)
- [Connecting to Databases](#)
- [Scripting](#)

In This Section ...

Writing SQL Queries

SQL Tables



For information about databases and how to get connected, see the [Database Connections](#) section.

The foundation of every database system is a table. Every database consists of one or more tables, which store the database's data/information. Each table is identified by a name (for example `CUSTOMERS` or `ORDERS`), and consists of column definitions and rows of data.

The database table columns have their own unique names and have pre-defined data types. Table columns can have various attributes defining the column functionality (such as the primary key, index, default value, and so on).

While table columns describe the data types, the table rows contain the actual data for the columns.

ID	Name	Address
1	Safeway	123 Safeway Lane
2	Costco	456 Costco Way
3	Target	789 Target Ave

Current common databases are almost all Relational Databases. This means that their tables can relate to each other by including an ID in one table that matches the key of another. These are called foreign keys.

Primary Key

A primary key is a way to uniquely identify each row in a table. While it is possible to create a database table without a primary key, it is highly recommended to configure one for each table. A primary key is comprised of either a single column, or set of columns. When multiple columns are specified as a primary key, this is known as a **composite primary key**. No two distinct rows in a table can have the same value (or combination of values) in those columns.

While Primary Keys can be configured in several ways, they *typically* meet the following criteria:

- **Integer Data Type:** The data type of the key column is typically an integer, and not a varchar. The primary key is only an identifier to a specific row in a table, so an integer data type can easily be used. Some databases support a UID or a UUID (Universally Unique Identifier) that looks like a character string, but is something specially made for primary keys.
- **Automatically Incrementing:** The value of the primary key increments as rows are added. The key is usually configured to automatically increment in the database so that external applications (such as Ignition) don't have to figure out the next available value when inserting a new row.
- **Statically Defined:** Any row that is inserted must fill in these value(s) without creating duplicates. Configuring the primary key as automatically incrementing means that the database will automatically handle this criteria.
- **Non-NUL:** NULL (empty) values should not be present in the primary key. This column (or columns) will usually not allow NULL values.

Index

Indexes speed up the querying process by providing swift access to rows in the data tables, similarly to the way a book's index helps you find information quickly within that book. Indexes are extremely important when querying large sets of data. You should create an index for the set of columns you use commonly in a `WHERE` clause. For example, you should add an index on the timestamp column of a historical table when querying the table by a start and end date. Ignition does this automatically when it creates tables for Tag History or Transaction Groups.

Foreign Key

A Foreign Key is a referential constraint between two tables. The foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table. The columns in the referencing table must be the primary key in the referenced table. For example, you might have a list of suppliers with an integer ID column. Then your invoices would use the supplier's ID instead of the name. These linked tables can save a lot of space because you don't have to include everything about the supplier in each invoice.

Example

Supplier (**SupplierNumber**, Name, Address, Type)
Invoices (InvoiceNumber, **SupplierNumber**, Text, Cost)

On this page ...

- [SQL Tables](#)
 - [Primary Key](#)
 - [Index](#)
 - [Foreign Key](#)
 - [Null Value](#)
 - [Comments](#)
- [SQL Queries](#)
 - [Select Command](#)
 - [Select Examples](#)
 - [Where Clause](#)
 - [Where Examples](#)
 - [Insert Into Command](#)
 - [Insert Examples](#)
 - [Update Command](#)
 - [Update Examples](#)
 - [Delete Command](#)
 - [Delete Examples](#)

Null Value

NULL is a special marker used in SQL to indicate that a data value does not exist in the database. This way it is clear that there is no data, instead of guessing if a value of 0 is correct or just missing data. By definition, NULL is not equal to anything, even other NULL values. Every database has a function to identify if a value is NULL, usually called `isNULL()` or something similar.

Comments

Comments can be added to any SQL query just like with scripting. Single line comments are done with two dashes and a space: '-- '

SQL - Single Line Comment

```
-- This is a single line comment in a SQL query.  
SELECT * FROM my_table
```

You can also do multi line comments by wrapping text within a forward slash and an asterisk: /* text */

SQL - Multi Line Comment

```
/* A multi line comment  
can span multiple lines. The  
comment will stop when it is closed  
with another asterisk and forward slash.*/
```

SQL Queries

SQL queries (or statements) are used to create, maintain, and view relational databases like MySQL, SQLServer, Oracle, etc. They follow a specific format and use just key words to determine the structure of the query. Unlike most coding languages, SQL does not rely on newlines or start/end markers for its format, each query is a single line of code. You will often see SQL queries split over several lines, but that is just to make them easier to read.



You might notice a lot of CAPITALIZED words in SQL queries. While these key words are not case sensitive, it is still common practice for people to capitalize them in a query. Things like `SELECT`, `FROM`, `WHERE` (and a few others) are almost always capitalized because they have a special meaning. Try not to have table or column names that use any of these special words. You will see this capitalization format in examples throughout this user manual and other online references.

Select Command

The `SELECT` statement is used to select data from a database. The result is returned as a data set, called the result set. This is true even if there is only one value returned. The syntax for a `SELECT` statement is as follows:

SQL - The Select Command

```
SELECT myColumn FROM myTable
```

Select Examples

The `**` character can be used to specify all columns from a database table. While this is the easiest way to retrieve results from a table, this is not the recommended approach.

SQL - Selecting All Columns

```
SELECT * FROM Customers
```

The recommended approach is to instead specify only the columns that are required for a query. There are several reasons for this, but performance would be the main one: less columns in a statement means less work for the database, and the resulting data set in Ignition will use less memory.

SQL - Selecting a Single Column

```
SELECT Name FROM Customers
```

SQL - Selecting Multiple Columns

```
SELECT Name, Address FROM Customers
```

Where Clause

The WHERE clause is used in conjunction with other commands to extract only those records that fulfill a specified criterion. The WHERE clause usually goes near the end of the query followed by a condition that the values must meet to be returned.

Pseudocode - Where Clause

```
SELECT myColumn FROM myTable WHERE condition
```

The WHERE clause can use various operators for its condition, with the basic operators being:

Operator	Description
=	Equal to.
<>	Not equal to.
>	Greater than.
<	Less than.
>=	Greater than or equal to.
<=	Less than or equal to.

Where Examples

Only return customers from CA.

SQL - Simple Where Clause

```
SELECT * FROM Customers WHERE State = 'CA'
```

Only return users over a specified age.

SQL - Select Users over 25

```
SELECT * FROM Users WHERE Age > 25
```

Insert Into Command

The INSERT INTO statement is used to insert a new row in a table. If any columns have default values or are auto-incrementing, they can be omitted from the INSERT query.

Pseudocode - The Insert Command

```
INSERT INTO myTable (column1, column2)
VALUES ('Value1', 'Value2')
```

Insert Examples

SQL - Insert Values into Columns

```
INSERT INTO Customers (Name, Address, City, State, Zip, Country, Phone)
VALUES ('Inductive Automation', '90 Blue Ravine', 'Folsom', 'CA', '95630', United States, '1-800-266-7798')
```

If inserting a value into every column of the table, the columns do not need to be listed on the INSERT INTO statement. The values just need to be listed in the same order as the columns in the table. The table in the query below has four columns: id, first name, last name, and title.

SQL - Inserting to all Columns

```
INSERT INTO Users
VALUES (5628, 'Bob', 'Smith', 'Project Manager')
```

Update Command

The UPDATE statement is used to update existing records in a table. If a WHERE clause is not used, **all rows in the table will be updated**. As a result, the UPDATE statement should be used in conjunction with a WHERE clause in most cases. Many official management tools like SQLServer's Management studio will not allow UPDATE commands without a WHERE clause.

Pseudocode - The Update Command

```
UPDATE myTable SET myColumn = 'myValue'
```

Update Examples

SQL - Updating All Rows in a Single Column

```
UPDATE Customers SET Name = 'Inductive Automation'
```

SQL - Updating a Single Column for a Single Row

```
UPDATE Customers SET Address = '2110 21st Street' WHERE ID = 1
```

Delete Command

The DELETE statement is used to delete records in a table. **NEVER** run a delete command without a WHERE clause. It will delete **ALL** records from that table. Many official management tools like SQLServer's Management studio will not allow DELETE commands without a WHERE clause.

Pseudocode - The Delete Command

```
DELETE FROM myTable WHERE myColumn = value
```

Delete Examples

SQL - Deleting Based on Column Value

```
DELETE FROM Customers WHERE Name = 'Inductive Automation'
```

SQL - Deleting Rows Based on an ID Column

```
DELETE FROM Customers WHERE id < 538
```

Related Topics ...

- [Inserting Data into a Database](#)
- [Named Queries](#)
- [Query Builder](#)
- [Database Query Browser](#)

In This Section ...

SQL Select Statements

While the SELECT command in its basic form can be very simple to use, the SELECT statement can be used with other statements or in certain ways that allow you to bring in exactly the data you need.

Selecting Static Values

Static values can be inserted into a resultset returned from a SELECT query as another column. Simply use the static value as a column to select, and the query will return a column where the name of the column is the static value, and every row in that column will return that same static value.

Pseudocode - Selecting Two Columns From a Table, and a Third Column with a Value of 10 for every row

```
SELECT column1, column2, 10 FROM table
```

On this page ...

- [Selecting Static Values](#)
- [Select Distinct Command](#)
- [Order By Clause](#)
- [Limiting Rows Returned](#)
- [Aliases](#)
- [Union Command](#)

Select Distinct Command

The SELECT DISTINCT statement works much like a SELECT statement works, in that it selects data from a database. However, SELECT DISTINCT will only return distinct or different values, not duplicates.

Pseudocode - The Select Distinct Command

```
SELECT DISTINCT column FROM table
```

Examples

This can be useful for getting a better idea of the range of values in a particular column.

SQL - Select Distinct Countries

```
SELECT DISTINCT country FROM Customers
```

Order By Clause

The ORDER BY keyword is used to sort the result-set by a specified column set of column. The ORDER BY keyword sorts the records in ascending (ASC) order by default. If you want to sort the records in a descending order, you can use the DESC keyword.

Pseudocode - Order By Clause

```
SELECT column1, column2 FROM table ORDER BY column2 DESC
```

Examples

SQL - Ordering by One Column

```
SELECT * FROM Customers ORDER BY Name ASC
```

You can use multiple columns to sort, this will sort by state first, and for each state the rows will be sorted by name.

SQL - Ordering by Multiple Columns

```
SELECT * FROM Customers ORDER BY State ASC, Name DESC
```

Limiting Rows Returned

SELECT commands can have the number of rows that the query returns limited using a special keyword. The keyword differs between database providers but the effect is the same, limiting the number of rows returned to a value that you specify.

Database	Keyword	Example
MS SQL Server/ MS Access	SELECT TOP value/percent	<p>Note that the SELECT TOP command is unique in that you can also specify a percentage value instead of an exact number of records.</p> <pre>SELECT TOP 200 column1, column2 FROM table</pre> <pre>SELECT TOP 10 PERCENT column1, column2 FROM table</pre>
MySQL	Limit value	<pre>SELECT column1, column2 FROM table LIMIT 200</pre>
Oracle DB	ROWNUM <= value	<p>Note that the ROWNUM command is unique in that you can use it to identify the row number for any reason, not just limiting the number of rows returned.</p> <pre>SELECT column1, column2 FROM table WHERE ROWNUM <= 200</pre>

Aliases

In a SQL query, aliases are used to give columns or even tables a temporary name for that query. Simply place the keyword AS after a column or table, followed by the alias name. If the alias is two words, it needs to be encapsulated in single quotes.

Pseudocode - Aliasing Columns and the Table

```
SELECT column1 AS a, column2 AS 'b c' FROM table AS t
```

Example

This can be really useful when the table has complex column names.

SQL - Aliasing Complex Column Names

```
SELECT id AS 'Badge Number', name AS 'Employee Name', dob AS Birthday FROM employees
```

This can also be useful when using multiple tables in a query, such as with a [JOIN](#).

SQL - Joining Columns from Two Tables with Aliases

```
SELECT * FROM Contacts AS co JOIN Customers AS cu ON cu.ID = co.CustomerID
```

Union Command

The UNION operator is used to combine the results of two different SELECT statements. This differs from a JOIN in that there does not have to be a relationship between columns. However, both SELECT statements need to select the same number of columns with similar data types in a similar order. So if my first statement selects an int column and then a string column, the second statement needs to do the same. The name of the columns in the resultset will take the name of the columns from the first SELECT in the UNION.

Pseudocode - Union Two Tables

```
SELECT stringCol, intCol FROM table1
UNION
SELECT stringCol, intCol FROM table2
```

Examples

By default, the UNION operator will only select distinct values between the two tables.

SQL - Union of Users and Customers

```
SELECT username FROM users
UNION
SELECT name FROM customers
```

To select all values from both tables, we can use UNION ALL instead.

SQL - Union All

```
SELECT jobTitle FROM jobs
UNION ALL
SELECT position FROM employees
```

Static values can be used in a UNION to help differentiate the rows from each table.

SQL - Differentiating Between Users and Customers

```
SELECT 'User' AS Type, username FROM users
UNION
SELECT 'Customer', name FROM customers
```

Related Topics ...

- [Writing SQL Queries](#)

SQL Where Clauses

Overview

The SQL WHERE clause is utilized to restrict the number of rows impacted by a query. A WHERE clause is commonly utilized in two scenarios:

- In conjunction with a SELECT statement to filter the number of rows returned.
- As part of an UPDATE or DELETE statements to restrict which rows are manipulated by the query.

In either scenario, the syntax of a WHERE clause is used the same, and can have multiple predicates:

Pseudocode - Where Clause Syntax

```
SELECT * FROM table WHERE column = value
```

On this page ...

- Overview
 - Where-Clauses and Data Manipulation
- And Operator
- Or Operator
- Not Condition
- Between Operator
- Like Condition
- In Condition
- Combining Multiple Operators

Where-Clauses and Data Manipulation

In most cases, statements that modify the content of a database table via UPDATE or DELETE **should** include a WHERE clause: otherwise the manipulation will be applied to every row. To demonstrate, you typically want to avoid queries that look like the following

Pseudocode - Never do this!

```
UPDATE table SET column = 'This was a horrible mistake'
```

When manually modifying a database table, a good habit to develop involves first writing a SELECT statement. If you can successfully write a WHERE clause that only returns the results you need to modify, then you can simply change the rest of your query to manipulate the table. Thus, we could write a query like the following:

Pseudocode - Select-Statement with a Where Clause

```
SELECT * FROM table WHERE id in (100,101,150,174)
```

If we receive only the results we need to modify in the query above, we can make a simple modification to our query to now delete just the rows we want.

Pseudocode - Delete-Statement with a Where Clause

```
DELETE FROM table WHERE id in (100,101,150,174)
```

Another common use of the WHERE clause is to search through records and return the ones during a particular time frame. Timestamp columns can use the < and > operators to compare to each other, and certain string formats can be compared to timestamps like this query:

Pseudocode - Where Clause with a Timestamp

```
SELECT * FROM table WHERE t_stamp > '1984-01-25 16:35:55'
```

And Operator

The AND operator allows you to specify two or more conditions in a WHERE clause, where each condition must be true for the row to be returned.

Pseudocode - Using And to Specify Multiple Conditions

```
SELECT column1, column2, column3 FROM table WHERE column1 > value AND column2 < value AND column3 = value
```

Example

This helps to narrow down the result set even further by adding in additional conditions that must be met. This will only return rows for customers from Germany who are also over 20 years old.

SQL - Customers from Germany Over 20

```
SELECT * FROM customers WHERE country = 'Germany' AND age > 20
```

Or Operator

The OR operator allows you to specify two or more conditions in a WHERE clause, but only one of the conditions need to be true for the row to be returned.

Pseudocode - Using Or to Specify Multiple Conditions

```
SELECT column1, column2, column3 FROM table WHERE column1 > value OR column2 < value OR column3 = value
```

Example

The OR operator can help pull in data from two different subsets in the table. This will only return rows for customers from Germany or customers who are over 20 years old.

SQL - Customers from Germany or Customers Over 20

```
SELECT * FROM customers WHERE country = 'Germany' OR age > 20
```

Not Condition

The NOT condition allows you to specify a condition that must not be met for the row to be returned.

Pseudocode - Using Not to Specify a Condition that Shouldn't be met

```
SELECT column1, column2, column3 FROM table WHERE NOT column1 = value
```

Example

This can be useful for finding all data other than a certain subset. This will return all customers who are not from Germany.

SQL - Customers that are not from Germany

```
SELECT * FROM customers WHERE NOT country = 'Germany'
```

Between Operator

The BETWEEN condition allows you to specify a range of values separated by an AND that the value must be in for a condition to be true. The value can be numbers, text or dates and is inclusive of the first and last values in the range.

Pseudocode - Using Between to Specify a Range of Values

```
SELECT column1, column2, column3 FROM table WHERE column1 BETWEEN value1 AND value2
```

Examples

SQL - Customers that are Between the Ages of 20 and 40

```
SELECT * FROM customers WHERE age BETWEEN 20 AND 40
```

Note that the BETWEEN operator would work similarly to using a greater than or equal condition and a less than or equal condition.

SQL - Customers that are Between the Ages of 20 and 40 with no Between Operator

```
SELECT * FROM customers WHERE age >= 20 AND age <= 40
```

Timestamps can also use the BETWEEN operator to check for a given start time and end time.

SQL - Customers that are Between the Ages of 20 and 40 with no Between Operator

```
SELECT * FROM customers WHERE start_time BETWEEN '1984-01-25 00:00:00' AND '1984-01-25 16:35:55'
```

Like Condition

The LIKE condition allows you to specify a condition that must meet a certain pattern. Typically used to compare to string values, the pattern can be built using a combination of characters and the two wildcard values.

- % - Used to specify any number of any characters including zero characters.
- _ - Used to specify exactly one character.

Pattern Examples

Pattern	Meaning	Possible Matches
'%a%'	Values that have an 'a' in them.	'a', 'Inductive Automation', 'almost', 'create'
'_a_'	Values that have an 'a' with exactly one character before and after the 'a'.	'bat', 'cat', 'can'
'_a%	Values that have an 'a' as the second character.	'da', 'saw', 'catcher'
'a%t'	Values that start with 'a' and end with 't'.	'about', 'at'
'%a%_%_%_'	Values that contain an 'a' with at least 3 other characters after it.	'trains', 'airplane', 'canteen'
'%a%a%'	Values that contain at least two 'a' characters in them.	'Inductive Automation', 'separate', 'apart'

Once the pattern has been constructed, it can be used with the LIKE operator to find values that match the specified pattern.

Pseudocode - Using Like to Specify a Pattern of Values

```
SELECT column1, column2, column3 FROM table WHERE column1 LIKE '%a%'
```

Example

The LIKE operator can be used to find all values that match a criteria, such as all countries with 'land' in that name

SQL - Customers that are not from Countries with 'land' in the Name

```
SELECT * FROM customers WHERE country LIKE '%land%'
```

In Condition

The IN operator allows you to specify a subset of values, with the condition that the return match at least one of them. Using an IN operator is similar to using multiple OR operators for the same column.

Pseudocode - Using IN to Specify Multiple Values

```
SELECT column1, column2, column3 FROM table WHERE column1 IN (value1, value2, value3)
```

Examples

The IN can be used as a shorthand way of writing out multiple conditions for the same column separated by OR operators. This would select all values where the country is either Germany, France, or USA.

SQL - Customers from Germany or France or USA

```
SELECT * FROM customers WHERE country IN ('Germany', 'France', 'USA')
```

This would be similar to doing something like this.

SQL - Customers from Germany or France or USA

```
SELECT * FROM customers WHERE country = 'Germany' OR country = 'France' OR country = 'USA'
```

The real power of the IN operator is that instead of specifying static values, an entirely new query can be run to compare values against.

SQL - Customers from Countries that users are also in

```
SELECT * FROM customers WHERE country IN (SELECT country FROM users)
```

Combining Multiple Operators

Multiple AND and OR operators can be combined to specify multiple different conditions that need to be met in order for a particular row to be returned. Additionally, each condition can be simple using the mathematical operators or complex using the conditions listed above. When using AND and OR operators in a WHERE clause, the AND will take precedence, evaluating first before the OR. In the pseudocode below, the row will be returned if either both the first and second conditions are met, or the third condition is met.

Pseudocode - Using Multiple Where Clause Operators

```
SELECT column1, column2, column3 FROM table WHERE column1 > value AND column2 < value OR column3 = value
```

However, the order at which the operators get evaluated can change by placing parentheses around the conditions which should be evaluated first. In the pseudocode below, the row will be returned if both the first condition is met, and either the second or third condition is met.

Pseudocode - Using Multiple Where Clause Operators with Parentheses

```
SELECT column1, column2, column3 FROM table WHERE column1 > value AND (column2 < value OR column3 = value)
```

Examples

We can use complex conditions with different operators to find all customers who are over the age of 50 in a country that has 'land' in the name, or any customers in Germany or France.

SQL - Multiple Complex Conditions

```
SELECT * FROM customers WHERE country LIKE '%land%' AND age > 50 OR country IN ('Germany', 'France')
```

Using parentheses in the same query can drastically change what valid return conditions are. Here, the customer must both be from a country with 'land' in the name, as well as either over 50 or from Germany or France.

SQL - Multiple Complex Conditions

```
SELECT * FROM customers WHERE country LIKE '%land%' AND (age > 50 OR country IN ('Germany', 'France'))
```

Related Topics ...

- [Writing SQL Queries](#)
- [system.db.runNamedQuery](#)

SQL Table Joins

Overview

The SQL JOIN allows you to run a single SELECT statement that references multiple tables. This can be used for more advanced filtering, as well as combining data from multiple tables in a single result set. The process of joining two tables involves stating both tables in the query, and then specifying that a column from one table relates to another in some way.

Joins may look imposing at first, but they are simply SELECT statements that utilize columns from multiple tables.

The JOIN keyword works in conjunction with a SELECT statement. However, there are some key concepts that must be addressed when attempting to use the JOIN keyword.

Specifying Each Column

When listing column in a statement that uses the JOIN keyword, you must denote which table each column is being retrieved from. You can do this by using an [Alias](#), or with the fully qualified column name. This prevents ambiguous columns in the context of the query, and makes it easier for you to use other keywords in the statement: i.e., adding WHERE clauses that apply to multiple tables.

Pseudocode - Fully Qualified Column Name

```
table_name.column_name
```

On this page ...

- Overview
 - Specifying Each Column
 - Declare the Relation
 - Combining the Concepts
 - Joins with Three or More Tables
- Join
 - Join in Action
 - Left Join
 - Left Join in Action
 - Right Join
 - Right Join in Action
 - Full Join
 - Full Join in Action
 - Full Joins in MySQL

Declare the Relation

After the JOIN keyword, you must state which columns from each table relate to each other. This is accomplished by stating the name of the table, using the ON keyword, and then stating that a column on the first table is equal to a column on the second table. The columns specified are typically [primary keys](#) for their respective tables.

In the example below, we're stating that the values in **some_column** on **tableA** should be associated with matching values in **some_other_column** on **tableB**.

Pseudocode - Using the Join Keyword

```
FROM
    TableA
JOIN tableB ON tableA.some_column = tableB.some_other_column
```

Note that the order you present the columns **after** the ON clause does not matter: A = B is equivalent to saying B = A, so we could switch the columns listed with no distinguishable impact on the resulting query.

Combining the Concepts

Altogether, a basic JOIN looks like the following:

Pseudocode - Join Syntax

```
SELECT
    tableA.column,
    tableB.column
FROM
    tableA
JOIN tableB ON tableA.identity_column = tableB.identity_column
```

Joins with Three or More Tables

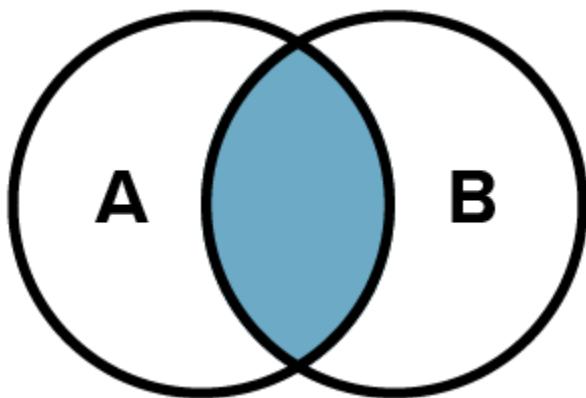
Joins can even be done between three or more tables. The syntax is similar, with each new table relation declared below the first.

Pseudocode - Join Syntax

```
SELECT
    tableA.column,
    tableB.column,
    tableC.column
FROM
    tableA
JOIN tableB ON tableA.identity_column = tableB.identity_column
JOIN tableC ON tableA.identity_column = tableC.identity_column
```

Join

The standard JOIN, also referred to as INNER JOIN, will only return rows where the joined columns contain matching values. If one of the joined columns contains a value that is not present in the other, then the row is not represented in the result set. You would use JOIN when you only want results that are represented in both tables.



This section will demonstrate the various uses of the JOIN keyword. For the sake of clarity, the queries will run against tables that look like the following:

Products Table	
id	product_name
1	Apples
2	Oranges
3	Grapes
4	Plums

Inventory Table			
id	product_id	quantity	product_vendor
1	1	15	Apple Corp
2	2	25	Orange Ya-Glad
3	3	56	Grape Escape
4	5	45	Banana Solutions

Join in Action

In this demonstration, only rows that pertain to Apples, Oranges, and Grapes are being returned. We're using a JOIN between **products.id** and **inventory.product_id**, so our results will only contain rows that have matching values from both of those columns. Our result set does not contain any information on products with **products.id** values of 4 or **inventory.product_id** values of 5, because those values are not present in **both** of the joined columns

SQL - Joining Products and Inventory

```

SELECT
    products.id
    ,inventory.product_id
    ,products.product_name
    ,inventory.product_vendor
    ,inventory.quantity
FROM
    products
JOIN inventory ON products.id = inventory.product_id

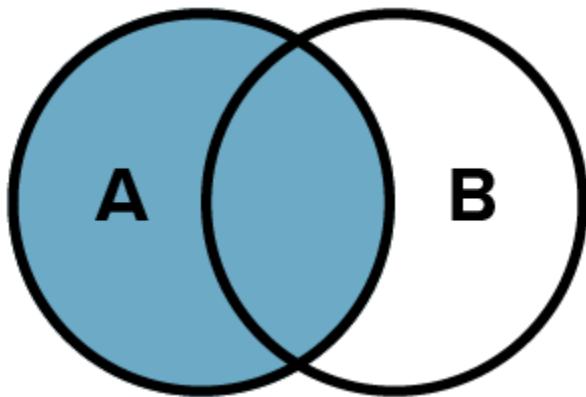
```

Example Results

id	product_id	product_name	product_vendor	quantity
1	1	Apples	Apple Corp	15
2	2	Orange	Orange Ya-Glad	25
3	3	Grapes	Grape Escape	56

Left Join

Return all rows from the left-most table (**table A** in the diagram), even if there are no matches on the right-most table (**table B**). If there isn't a matching record in the right table, then NULL values are returned.



Left Join in Action

Here we see all rows returned from our products table (since it is the left-most table in our query). In row 4, columns that are being populated via the inventory table (**product_id**, **product_vendor**, and **quantity**) contain NULL values, because there isn't a row on the inventory table that matches with a **product_id** value of 4. The query must return something in this case, so it returns NULL for these columns.

SQL - Left Joining Products and Inventory

```

SELECT
    products.id
    ,inventory.product_id
    ,products.product_name
    ,inventory.product_vendor
    ,inventory.quantity
FROM
    products
LEFT JOIN inventory ON products.id = inventory.product_id

```

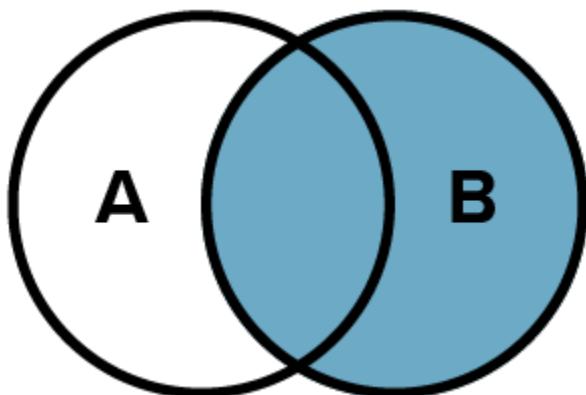
Example Results

id	product_id	product_name	product_vendor	quantity
1	1	Apples	Apple Corp	15

1	1	Apples	Apple Corp	15
2	2	Orange	Orange Ya-Glad	25
3	3	Grapes	Grape Escape	56
4	NULL	Plums	NULL	NULL

Right Join

Return all rows from the right-most table (**table B**), even if there are no matches on the left-most table (**table A**). If there isn't a matching record on the left table, then NULL values are returned.



Right Join in Action

When using a RIGHT JOIN, all rows will be returned from the inventory table. The products table does not have a row that contains an **id** value of 5, so the **id** and **product_name** columns will show NULL values in our result set.

SQL - Right Joining Products and Inventory

```

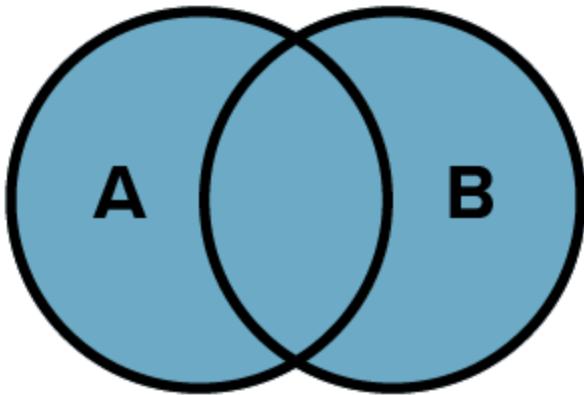
SELECT
    products.id
    ,inventory.product_id
    ,products.product_name
    ,inventory.product_vendor
    ,inventory.quantity
FROM
    products
RIGHT JOIN inventory ON products.id = inventory.product_id
  
```

Example Results

id	product_id	product_name	product_vendor	quantity
1	1	Apples	Apple Corp	15
2	2	Orange	Orange Ya-Glad	25
3	3	Grapes	Grape Escape	56
NULL	5	NULL	Banana Solutions	45

Full Join

The FULL JOIN returns all rows from both tables, regardless if there are matching values in the joined columns. You would use a FULL JOIN in cases where you want to show all applicable records from both tables, and synchronize the data across both tables via the joining columns where possible.



Full Join in Action

Note that we're using the same query as the standard JOIN, but we've prepended "FULL" to the last line of the query. Note the NULL values returned in cases where our product ID is not fully represented on both tables.

SQL - Full Joining Products and Inventory

```

SELECT
    products.id
    ,inventory.product_id
    ,products.product_name
    ,inventory.product_vendor
    ,inventory.quantity
FROM
    products
FULL JOIN inventory ON products.id = inventory.product_id
  
```

Example Results

id	product_id	product_name	product_vendor	quantity
1	1	Apples	Apple Corp	15
2	2	Orange	Orange Ya-Glad	25
3	3	Grapes	Grape Escape	56
4	NULL	Plums	NULL	NULL
NULL	5	NULL	Banana Solutions	45

Full Joins in MySQL

MySQL does not have an equivalent FULL JOIN. However, you can emulate one by utilizing a LEFT JOIN, RIGHT JOIN, and the UNION keyword. For the sake of simplicity, we will return all columns in the following example, but you would still want to specify individual columns in both SELECT query.

SQL - Full Join in MySQL using Left Join, Right Join, and Union Keyword

```

SELECT * FROM products
LEFT JOIN inventory ON products.id = inventory.product_id

UNION ALL

SELECT * FROM products
RIGHT JOIN inventory ON products.id = inventory.product_id
WHERE products.id IS NULL
  
```

Related Topics ...

- [Writing SQL Queries](#)
- [SQL Query Data Source](#)

SQL Common Functions

Functions are available in most SQL databases, and can provide some helpful utility to any queries you may be calling in Ignition.

This page contains some commonly used SQL functions that some databases contain. The exact functions available and usage depends on the database, so always check your database's documentation for a more complete list of available functions.

Using Column Values

It is important to understand that when calling these functions, you generally use a column name instead of a static number. For the sake of simplicity, the tables below demonstrate how to use the functions with static values, but they usually are switched for column names, ie:

Pseudocode - Passing a Column to a Function

```
SELECT SUM(downtime_duration) FROM downtime_events
```

On this page ...

- Using Column Values
 - Example Table
- Numeric Functions
- String Functions
- Date Functions
- Logic Functions
- Group By Clause

Example Table

Some of the functions on this table are better demonstrated when used in conjunction with a table (i.e., using the AVG() function with a single value isn't too interesting). Thus, the following table contains sample data that the functions on this page will utilize if necessary.

Products Table

id	product_quantity	product_name	date_added	date_updated	origin_state
1	100	apples	Mon Jan 29 00:00:00 PST 2018	Mon Jan 29 12:00:00 PST 2018	California
2	24	oranges	Mon Feb 13 00:00:00 PST 2017	Mon Feb 13 09:00:00 PST 2017	Florida
3	56	grapes	Mon Mar 07 00:00:00 PST 2016	Mon Mar 07 05:00:00 PST 2016	California

Numeric Functions

Function	Description	Example	Output
ABS(value)	Returns the absolute value of the passed number or column.	<pre>SELECT ABS (-3.5)</pre>	3.5
AVG(value)	Takes the values of a single numeric column, and returns an average. A WHERE clause may be used in the same statement to filter out some of the rows on the table.	<pre>SELECT AVG (product_quantity) FROM products</pre>	60
CEILING(value)	Returns the next greatest integer value based on the argument provided. Thus, CEILING(10.1) would return 11.	<pre>SELECT CEILING(10.1)</pre>	11

COUNT(value)	Returns a row count. Typically takes either a single column, *, or 1. Regardless of which row is passed, the function will return the number of rows on the table that meet the criteria of any WHERE clauses.	<pre>SELECT COUNT (*) FROM products</pre>	
FLOOR(value)	Returns the next smallest integer value based on the argument provided. Thus, FLOOR(10.9) would return 10.	<pre>SELECT FLOOR (10.9)</pre>	10
MAX(value)	Returns the largest value from the specified column.	<pre>SELECT MAX (product_quantity) FROM products</pre>	100
MIN(value)	Returns the smallest value from the specified column.	<pre>SELECT MIN (product_quantity) FROM products</pre>	24
ROUND(value, decimal_places)	Returns a number rounded to a certain number of decimal places. Takes two parameters. The first is the number to round to, and the second is the number of decimal places to round to.	<pre>SELECT ROUND (1.234, 1)</pre>	1.2
SUM(value)	Takes the value of a single numeric column, and returns the sum. A WHERE clause may be used in the same statement to filter out some of the rows on the table.	<pre>SELECT SUM (product_quantity) FROM products</pre>	180

String Functions

String Functions		Example	Output
CONCAT (value1, value2,..., valueN)	Concatenates multiple strings or values. Some databases may require you to convert each value to a string before concatenating.	<pre>SELECT CONCAT (product_name, ':', product_quantity) FROM products</pre>	apples:100 oranges:24 grapes:56

LOWER (value)	Converts a string to lowercase.	<pre>SELECT LOWER('MAKE Me smALL')</pre>	make me small
LTRIM(value)	Removes leading space from a string.	<pre>SELECT LTRIM(' Take a little off the left')</pre>	Take a little off the left
REPLACE (original_string, target_string, replacement_string)	Searches a string for a substring (target_string), and replaces the substring with the replacement_string.	<pre>SELECT REPLACE('Who is awesome', 'Who is', 'You are')</pre>	You are awesome
RTRIM(value)	Removes leading space from a string	<pre>SELECT RTRIM('Take a little off the right ')</pre>	Take a little off the right
SUBSTRING (original_string, character_index, [length])	Extracts a substring from another string based on character index. Takes two parameters: the original string, and the character index to start at. An optional third parameter can specify the number of characters to extract. Character index is one-based, so the first character in the string rests at index 1.	<pre>SELECT SUBSTRING('This is my string!', 9, 9)</pre>	my string
TRIM(value)	Removes both leading and trailing space from a string.	<pre>SELECT TRIM(' Trim Both Sides ')</pre>	Trim Both Sides
UPPER(value)	Converts a string to uppercase.	<pre>SELECT UPPER('super size me')</pre>	SUPER SIZE ME

Date Functions

There are many date and time functions for each database (MySQL, MSSQL, Oracle, etc), but they all vary wildly. These examples work in most databases:

Function	Description	Example	Output
CURRENT_TIMESTAMP()	Returns the current date and time, as reported by the database.	<pre>SELECT CURRENT_TIMESTAMP()</pre>	Returns the current time
TIMEDIFF(date1, date2)	Returns a difference between two dates. Assumes that date1 is the most recent datetime.	<pre>SELECT TIMEDIFF (date_updated, date_added) FROM products WHERE id =1</pre>	Thu Jan 01 12:00: 00 PST 1970

Logic Functions

Function	Description	Example	Output
COALESCE (value1, value2,... valueN)	Returns the first non-null expression.	<pre>SELECT COALESCE (NULL, 'Pick me!')</pre>	Pick me!
ISNULL (expression)	Returns true if an expression is NULL.	<pre>SELECT ISNULL (NULL)</pre> <pre>SELECT ISNULL (14)</pre>	True False
NULLIF (expression1, expression2)	Compares two expressions. If they are equal to each other, then the function returns a NULL. If the two expressions are not equal, the first expression passed to NULLIF() is returned.	<pre>SELECT NULLIF (100, 100)</pre> <pre>SELECT NULLIF (100, 3)</pre>	NULL 100

Group By Clause

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns. This way you can find the MIN, MAX, Average, COUNT, etc., for each group of rows. Grouping can also be done for multiple columns, with precedence going in the order that they are listed.

Pseudocode - Passing a Column to a Function

```
SELECT SUM(column1) FROM table GROUP BY column2
```

Example

Here, we are grabbing the sum of product quantity for each origin state.

SQL - Simple Group By

```
SELECT SUM(product_quantity), origin_state, FROM products_table GROUP BY origin_state
```

Related Topics ...

- [Writing SQL Queries](#)

- SQL Query Data Source
- system.db.runNamedQuery

SQL Stored Procedures

For the uninitiated, Stored Procedures are a series of predefined SQL statements that are configured and stored in the database, and can be easily called from another application, such as Ignition, with a single statement. Conceptually, they are very similar to a scripting function: parameters may be passed to them, they can utilize looping and conditional logic, and can return a value.

Stored Procedure Syntax

Stored procedures are created and maintained in your database. As a result, the creation of a stored procedure is outside the scope of the Ignition User Manual: the commands used to create a stored procedure vary per database.

In regard to calling a Stored Procedure, the syntax can also differ.

SQL Server

The EXEC command is utilized to execute a stored procedure.

Pseudocode - Executing a Stored Procedure in SQL Server

```
EXEC dbo.myStoredProcedure
```

Parameters may be passed to the Stored Procedure. SQL Server's documentation has more details on utilizing parameters with Stored Procedures.

Pseudocode - Executing a Stored Procedure in SQL Server with Parameters

```
EXEC dbo.myStoredProcedure @myParam = 100, @AnotherParameters = 'Hello'
```

MySQL

MySQL uses the CALL command to execute a Stored Procedure. Note the parentheses characters at the end, as they must be present even when the Stored Procedure is not utilizing parameters.

Pseudocode - Executing a Stored Procedure in MySQL

```
CALL myStoredProcedure()
```

If parameters are defined, they can be passed via the parenthesis, similar to how functions work in Python.

Pseudocode - Executing a Stored Procedure in MySQL with Multiple Parameters

```
CALL myStoredProcedure(100, 'Hello')
```

For information on the creation of a stored procedure, as well as proper SQL syntax to call a Stored Procedure, reference your database's documentation. Alternatively, if you have a database administrator, they can typically help with the creation and execution of a Stored Procedure.

Calling Stored Procedures in Ignition

There are several locations in Ignition where Stored Procedures may be utilized from.

SQL Query Bindings

Instead of typing a query directly into a [SQL Query binding](#), a Stored Procedure may be executed instead. Assuming a MySQL database contains a Stored Procedure named 'return_all_bays', we can call the procedure on a binding with the following:

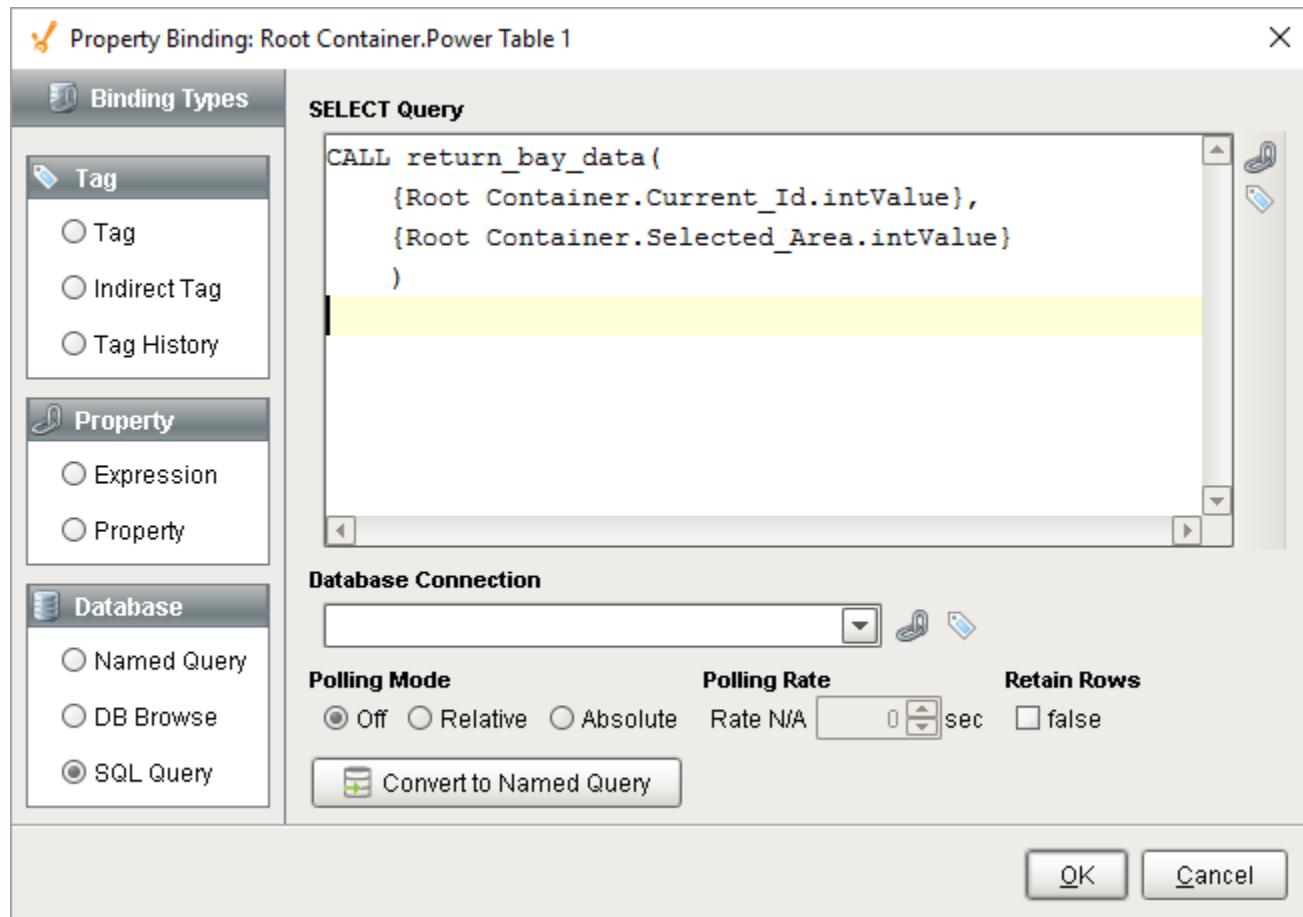
On this page ...

- [Stored Procedure Syntax](#)
 - [SQL Server](#)
 - [MySQL](#)
- [Calling Stored Procedures in Ignition](#)
 - [SQL Query Bindings](#)
 - [Named Queries](#)
- [Stored Procedure Groups](#)
- [Stored Procedures in Python Scripts](#)
 - [Using createSProcCall and execSProcCall](#)
 - [Other System Functions](#)

SQL - Calling a Stored Procedure in MySQL

```
CALL return_all_bays()
```

As with all bindings, Tag  and Property  references may be utilized by clicking the appropriate icons in the binding window.



Named Queries

Named Queries support Stored Procedure calls. As mentioned under [Stored Procedure Syntax](#), the syntax is based entirely on how your database expects a Stored Procedure to be called. Be mindful of the [Query Type setting](#), as it has to match what the stored procedure is doing: if it is returning a result set, leave it set to Query, if it is modifying a record in the database, then set the type to Insert Query.

The screenshot shows the Ignition Project Browser interface. On the left, the Project Browser tree includes nodes like Alarm Notification Pipelines, Sequential Function Charts, Scripting, Perspective, Transaction Groups, Vision, Named Queries (with sub-nodes paramProblems, Inventory, MyNamedQuery, OEEValues, Query), Stored Procedure (selected), Update Query, and Reports.

The main workspace is titled "Stored Procedure" and has three tabs: Settings, Authoring (selected), and Testing. Under "Authoring", there are sections for Database Connection (set to <Default>) and Query Type (set to Query). The Parameters table lists two parameters: firstParam (Type: Value, Data Type: Int4) and secondParam (Type: Value, Data Type: String). The Query section contains the SQL command:

```

1 CALL myStoredProcedure( :firstParam, :secondParam)
2

```

Stored Procedure Groups

One of the easiest ways to utilize Tags with a Stored Procedure is to use the [Stored Procedure Group](#). Parameters can be easily assigned to each item in the group, and utilize all of the features of a [Transaction Group](#), such as scheduled execution and triggers.

Each item in the group is linked to a specific parameter in the Stored Procedure. Any IN or INOUT parameters can write directly to the Tags, while new values can be fed into OUT and INOUT parameters allowing you to easily move data from Tags into the database with the Stored Procedure.

The screenshot shows the configuration interface for a Stored Procedure Group. The left side displays three tables:

- Basic OPC/Group Items (1)**: Contains one item: Daily Total ... (Value: 100, Target Name: passBack, Data Type: Int4).
- Run-Always Expression Items (ignore trigger) (0)**: Empty table.
- Triggered Expression Items (0)**: Empty table.

The right side of the interface includes several configuration tabs:

- Action**: Shows Execution Scheduling (Timer selected, 1 second(s)), Data source (<Default>), and Procedure name (retrieve_daily_total).
- Trigger**: Includes options for Store timestamp to and Store quality code to.
- Options**: Not fully visible in the screenshot.

Stored Procedures in Python Scripts

There are a few ways to call a Stored Procedure from a script in Ignition.

Using `createSProcCall` and `execSProcCall`

The recommended approach to calling a Stored Procedure from a Python script in Ignition typically involves two main steps:

1. Calling [system.db.createSProcCall](#) to create a call context, or object that effectively represents the impending stored procedure call. This object can be used to specify parameter values that will be passed to the Stored Procedure.
2. Using [system.db.execSProcCall](#) to execute the Stored Procedure.

Once the Stored Procedure has been executed, the call context generated in step #1 can be used to reference any values that were returned.

Other System Functions

Technically, most other system functions in the "db" library, such as [system.db.runPrepQuery](#), can be used to call a Stored Procedure. We generally recommend against this, as [system.db.createSProcCall](#) and [system.db.execSProcCall](#) are better suited to work with Stored Procedures and have some additional functionality not found in the other db functions.

Related Topics ...

- [SQL Query Bindings in Vision](#)
- [Understanding Transaction Groups](#)

Query Builder

Crafting Queries with the Query Builder

Many places in Ignition that allow for SQL queries have a link to the SQL Query Builder tool. The Query Builder is a powerful Drag-and-Drop query building GUI that allows you to make complex queries from your connected databases. While a basic understanding of SQL helps make the most of the Query Builder tool, most people will have no problem creating effective queries after a brief tutorial. Additionally, the Query Builder does go over many advanced features of SQL that may be unfamiliar. We suggest looking up how these work in your favorite SQL resource guide as this covers how to use them, not what they do.

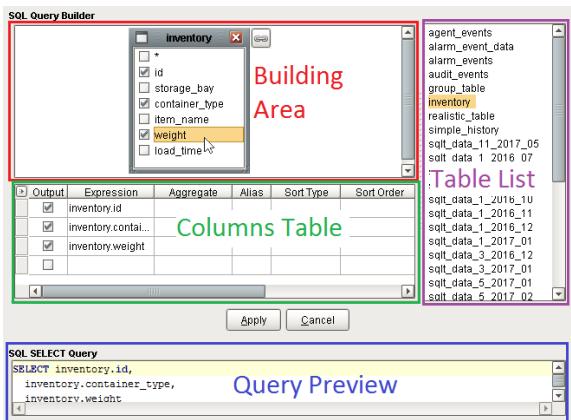
Builder Anatomy

Once opened, the Query Builder has the following items:

- **Building Area:** Visualizes the table relationships of all tables included in the query. Tables are typically dragged in from the **Table List**.
- **Columns Table:** Shows which columns from all tables are referenced in the query.
- **Table List:** Shows all database tables in the selected Database Connection.
- **Query Preview:** Shows a preview of the query that will be created once the **Apply** button has been pressed.

On this page ...

- [Crafting Queries with the Query Builder](#)
 - [Builder Anatomy](#)
 - [Opening the Query Builder](#)
 - [Using the Syntax Parser](#)
 - [Using the Builder](#)
 - [Query Properties](#)
- [Joining Tables in the Query Builder](#)
 - [Joins Right-Click Menu](#)
 - [Join Properties](#)
- [Columns Table](#)
 - [Right-Clicking on an Item](#)
 - [Field Description](#)



Opening the Query Builder

The Builder can be accessed from several different resources in Ignition, notably when using [Named Queries](#) and setting up [Report Data](#). To use the Builder, click the **Show Builder** button.

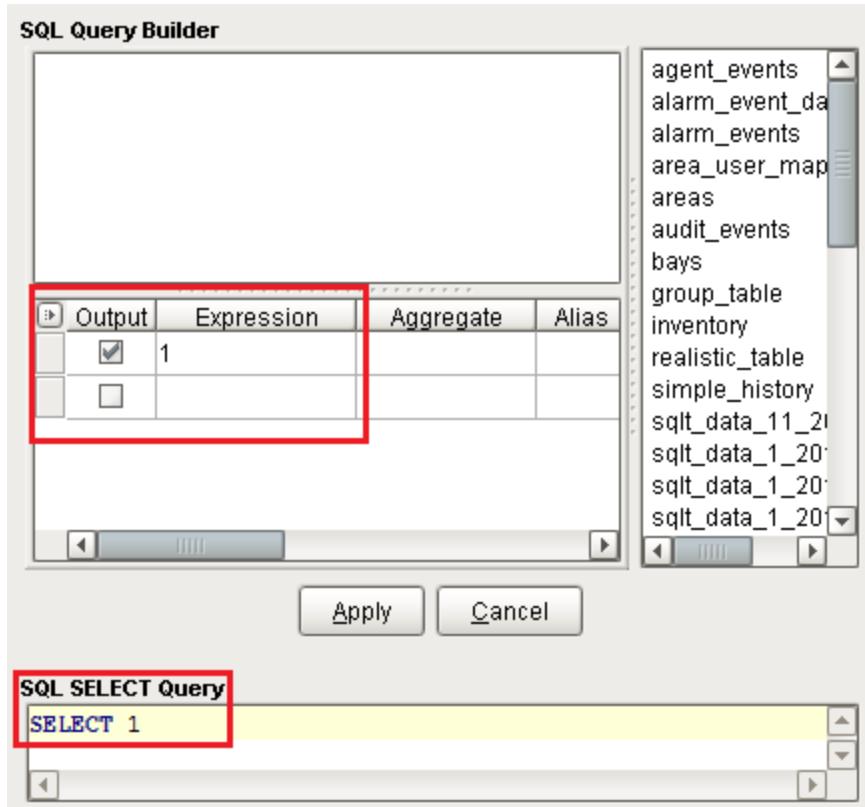


Clicking this button will open the Builder workspace with an empty query. If a query was already written before the button was pressed, then the Builder will attempt to load the query into the Builder's interface. For example, if the following query was typed before pressing the button:

SQL - Select Statement - Loading into the Builder's Workspace

```
SELECT 1
```

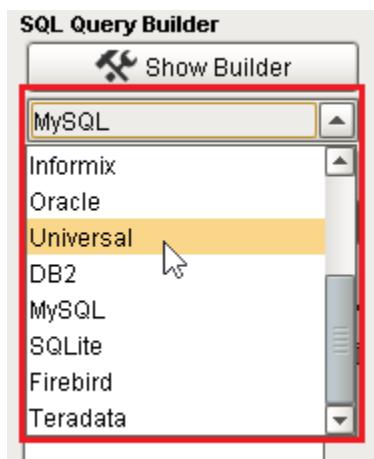
The following would be loaded into the Builder's workspace. Note that the **Building Area** is empty, because we're not querying from a table.



The Builder attempts to parse your query based on the **Syntax Parser**.

Using the Syntax Parser

When starting with a pre-existing query, the Syntax Parser tells the Builder which implementation of SQL your query's syntax is using. In most cases, this can be left with the default value of **Universal**. When set to a specific implementation of SQL, this allows the Builder to understand/accept implementation-specific keywords and syntax.



Syntax Parser in Action

In MySQL, query results may be limited with the **LIMIT** keyword:

```
SQL - Limit Statement

SELECT
    *
FROM
    myTable
```

```
LIMIT  
    100
```

Attempting to open the Builder while using **LIMIT**, and the Syntax Parser is configured to a syntax that does not recognize the **LIMIT** keyword, will result in an error:

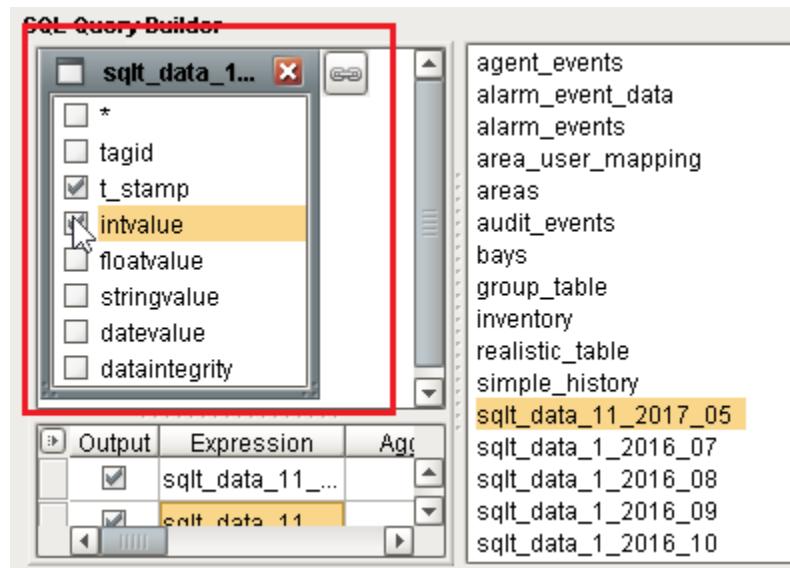


In these scenarios, you can switch to a syntax that supports the query, or remove the offending lines.

Using the Builder

Once the Builder is open, (and if you have tables in your database connection), you can start building a query by dragging and dropping a table from the Table List into the Building Area. From here, you can select which columns on the Table Object you want to bring into the query, or drag and drop them into the Columns Table below.

The * option at the top of the table is special. If the * character is selected, then all columns from the table will be included in the results just like a "SELECT * " query.



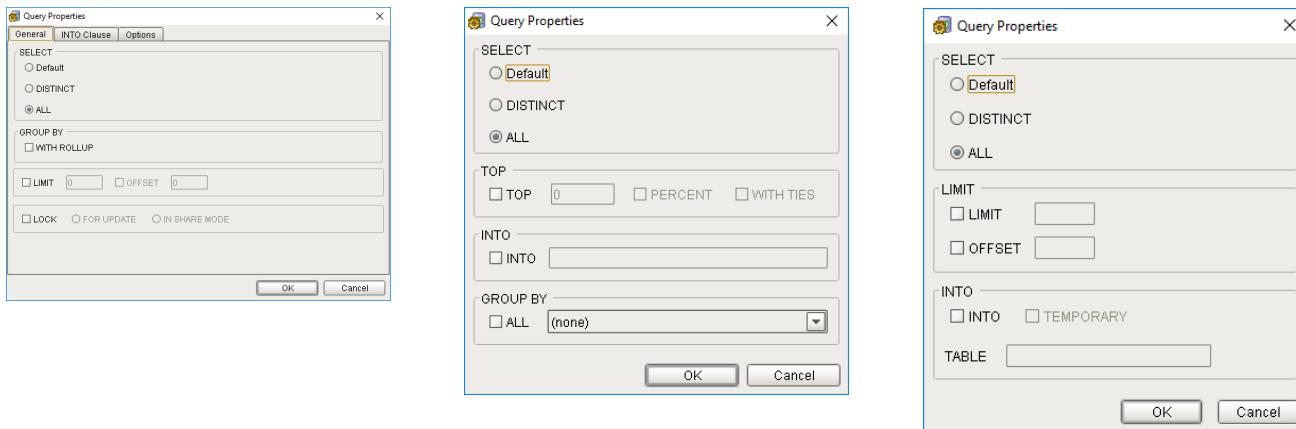
Query Properties

Right clicking on the empty space in the Building Area and selecting properties will bring up the Query Properties window that allows you to customize how the query works. How it looks and what it contains can vary, depending on what syntax parser you have selected.

[MySQL Query Properties](#)

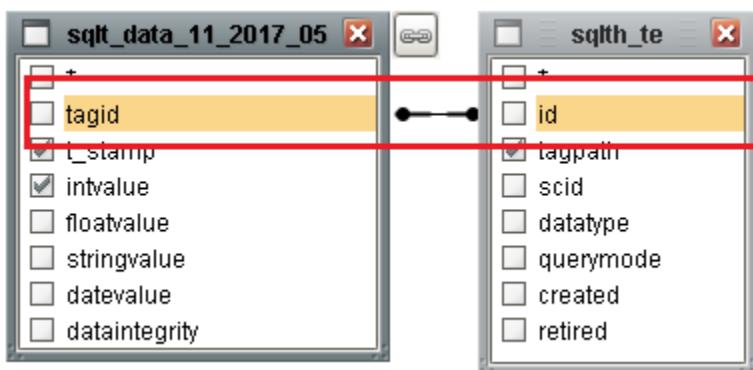
[MSSQL Query Properties](#)

[PostgreSQL Query Properties](#)



Joining Tables in the Query Builder

Multiple tables may be added to the Building Area. Once there, the Builder can JOIN the two tables by dragging from one column in a table on to another column on a different table.

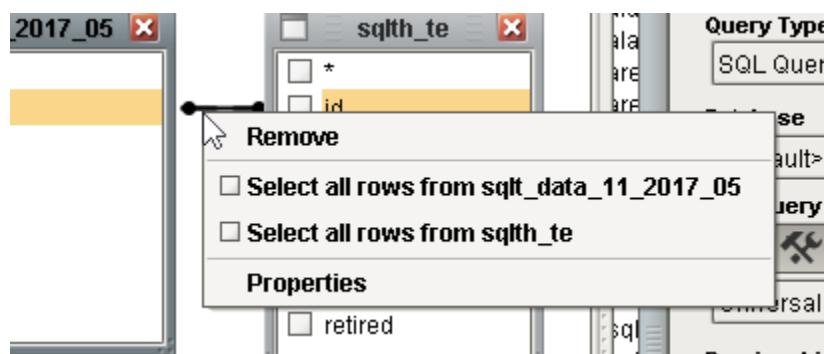


Notice that "tagid" and "id" are now linked together. This means the resulting query will JOIN the two tables based on the linked columns. Much like a SQL JOIN, the joining columns do not need to be selected to be utilized in the JOIN.

Joins Right-Click Menu

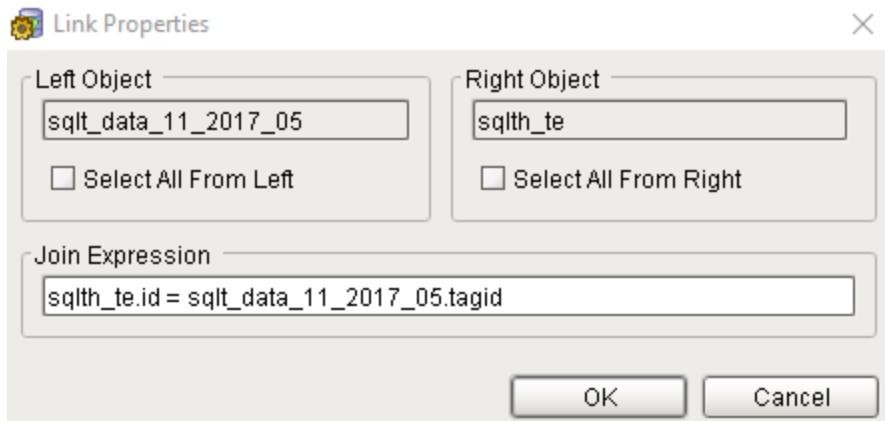
Right-clicking on the link will create a menu that allows you to remove the JOIN, as well as access properties of the JOIN.

Additionally, you can select all rows from one of the joining tables, which effectively creates a **RIGHT OUTER JOIN** or **LEFT OUTER JOIN**, depending on which table you select.



Join Properties

Clicking on **Properties** on the Right-Click Menu allows you to view the relation between each table, as well as apply **LEFT** or **RIGHT JOINs**, as mentioned above.



Columns Table

The Columns Table allows you to modify individual columns in the query. Typically, this allows you to add aggregates or aliases to each column from the Builder. Each row represents an expression, or combination of functions, columns, variables, and constants that will ultimately be a single column in the resulting query.

Output	Expression	Aggregate	Alias	Sort Type	Sort Order
<input checked="" type="checkbox"/>	sqlth_te.tagpath				
<input checked="" type="checkbox"/>	sqlt_data_11_2...				
<input checked="" type="checkbox"/>	sqlt_data_11_2...				
<input type="checkbox"/>					

Right-Clicking on an Item

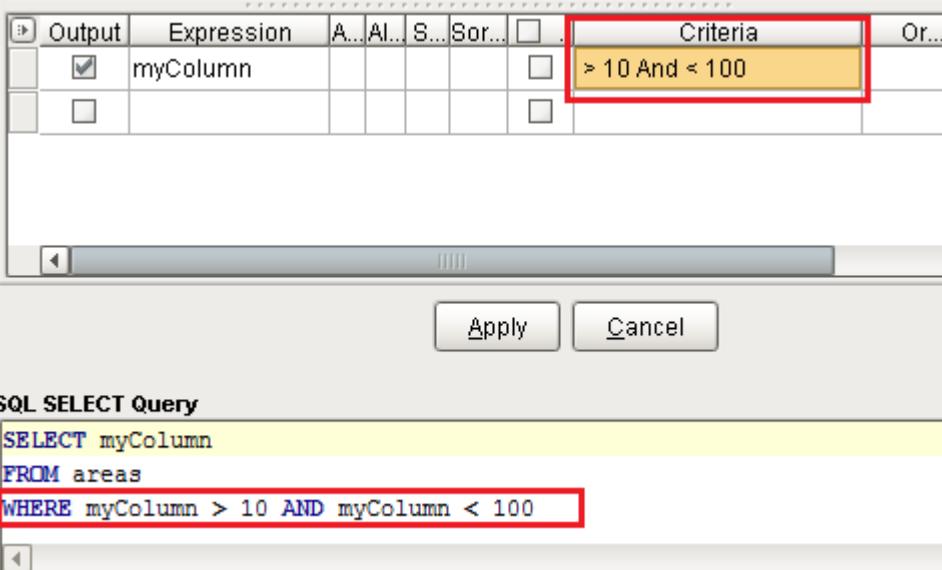
Right-clicking on a row of the Columns Table brings up a menu with the following options:

Option	Description
Move up	Moves the selected row up in the table. This means the column will appear in the query results prior to the other listed columns.
Move down	Similar to move up, but moves the row down, or towards the end of the listed columns.
Delete item	Removes the row from the Columns Table, effectively removing it from the resulting query.
Insert empty item	Adds a new row to the Columns Table with blank values.

Field Description

Each column in the Columns Table allows you to modify the resulting query in some way. The following is a description of each column:

Name	Description
Output	Specifies if the row should appear in the resulting query. Disabling a row on the table removes the column from the results.
Expression	The column from one of the tables in the Building Area that this row represents. Left-clicking on this field will create a dropdown list of possible rows.
Aggreg	

ate	<p>Allows you to aggregate the expression. Blank by default (no aggregate), configuring this column allows you to do things like sum column, or return the average. Possible values are:</p> <ul style="list-style-type: none"> • Blank (no aggregation will be performed) • Avg • Count • Max • Min • Sum <p>You can also add the Distinct keyword to any non-blank option. For example, you could enter "Sum Distinct" or "Count Distinct".</p>
Alias	Creates an Alias, or alternative name for the column. Effectively adds the SQL "AS" keyword to the column, which allows you to rename the resulting column.
Sort Type	Sorts the results based on this column. Enabling a Sort Type is similar to added an ORDER BY to your query. Possible values are: <ul style="list-style-type: none"> • Blank (no sorting on this column will be performed) • Ascending • Descending
Sort Order	When a Sort Type has been specified on multiple rows, this determines which row the query will sort on first.
Grouping	Allows you to group the results. Adds a GROUP BY statement to the resulting query. Enabling grouping on one of the columns makes the Criteria for column appear.
Criteria for	<p>A dropdown list that sets how you want the Criteria column to apply to the grouping. Does not apply if both Grouping and the WHERE clause Criteria are not being used.</p> <ul style="list-style-type: none"> • For Values will place the WHERE clause specified in the Criteria column first, filtering out rows that don't apply, and then applying the grouping. • For Groups will instead change the WHERE clause into a HAVING clause that is used to filter after the grouping has taken place.
Criteria	<p>Allows you to add a WHERE clause. Supports the use of both the OR and AND keywords for multiple conditions. Example: assuming a column named "myColumn", we could limit the results of our query to rows where myColumn has a value greater than 10 AND less than 100:</p>  <pre> SQL SELECT Query SELECT myColumn FROM areas WHERE myColumn > 10 AND myColumn < 100 </pre>
Or...	Allows you to add an additional condition to a WHERE clause. Will separate each grouped condition with "(" characters to maintain logic.

Note: The Query Builder is a third party tool that we brought into Ignition and while we go over how to use it here, you can also check out the [Active Query Builder's documentation](#) for more information on how this feature works.

Related Topics ...

- [Named Queries](#)

- [Named Query Data Source](#)

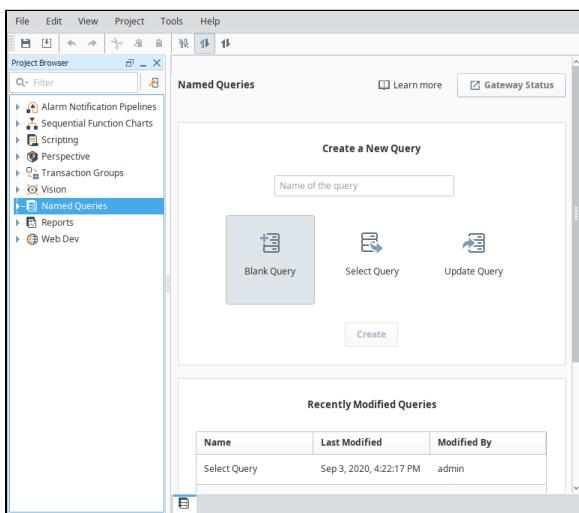
Named Queries

What Are Named Queries?

Named Queries are preconfigured queries that may be invoked elsewhere in a project. Named Queries are conceptually very similar to Project Scripts; defined in a single location, and then referenced in multiple places throughout the project. When executing a Named Query, parameters may be passed to return a dynamic result set. This way, a query may be written once, and then called from multiple locations throughout the project.

The Named Queries Welcome tab allows you to create three types of named queries. Each one of the named query types is basically a template to help you get started creating your own query. Once you select a Named Query type, enter a name, and press 'create', and the specific named query template will open. The Select Query and the Update Query will have some sample parameters and queries to help you get started. The Named Queries Welcome tab will show you any recently modified named queries along with the date it was modified and who modified it. You can even double click on a recently modified query and open it.

The Named Queries Welcome tab provides a quick way to create a new query and update existing ones.



Named Queries have their own [workspace](#) in the Project Browser section of the Designer.

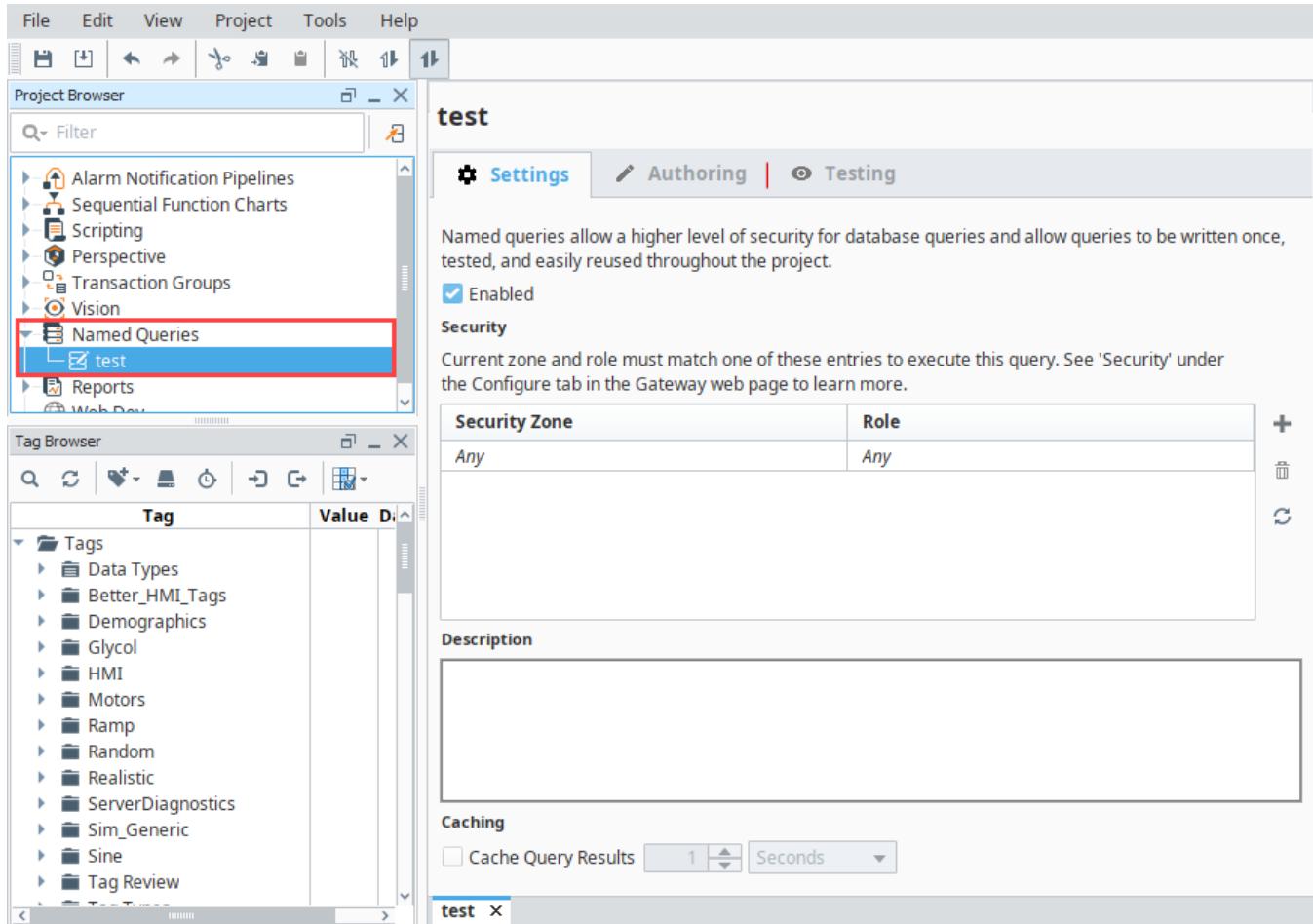
On this page ...

- [What Are Named Queries?](#)
- [Secure Query Execution](#)
- [Where Are Named Queries Used?](#)
 - [Binding](#)
 - [Reporting](#)
 - [Scripting](#)



Named Query Overview

[Watch the Video](#)



Secure Query Execution

While clients may request data from a Named Query, the actual execution of the query always takes place on the Gateway. Clients simply specify which query should run, and pass parameters that the Gateway will use. Additionally, the Gateway has an opportunity to [restrict access](#) to the query based on Security Zone and/or User Role. This provides a single interface to restrict access to the queries, and better protect your data. Additionally, queries cannot be modified by a Client other than by passing variables into it. This creates a very secure method to control what queries are being run against your database.

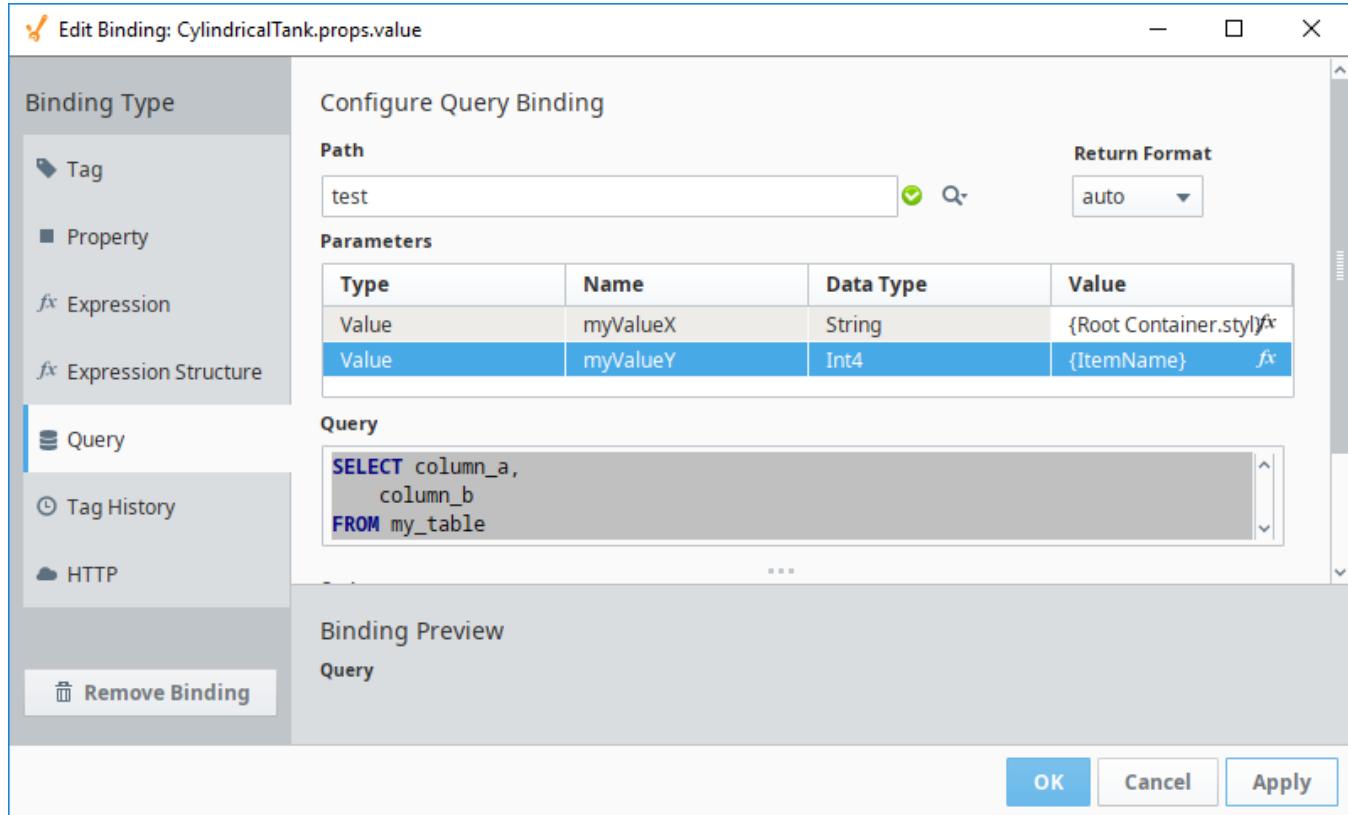
Where Are Named Queries Used?

Named Queries may be used in multiple locations in Ignition. They are used in all the same places that a normal SQL query would be used. Here are a few examples of locations in which named queries can be used. A more detailed example of a named query is provided [here](#).

Binding

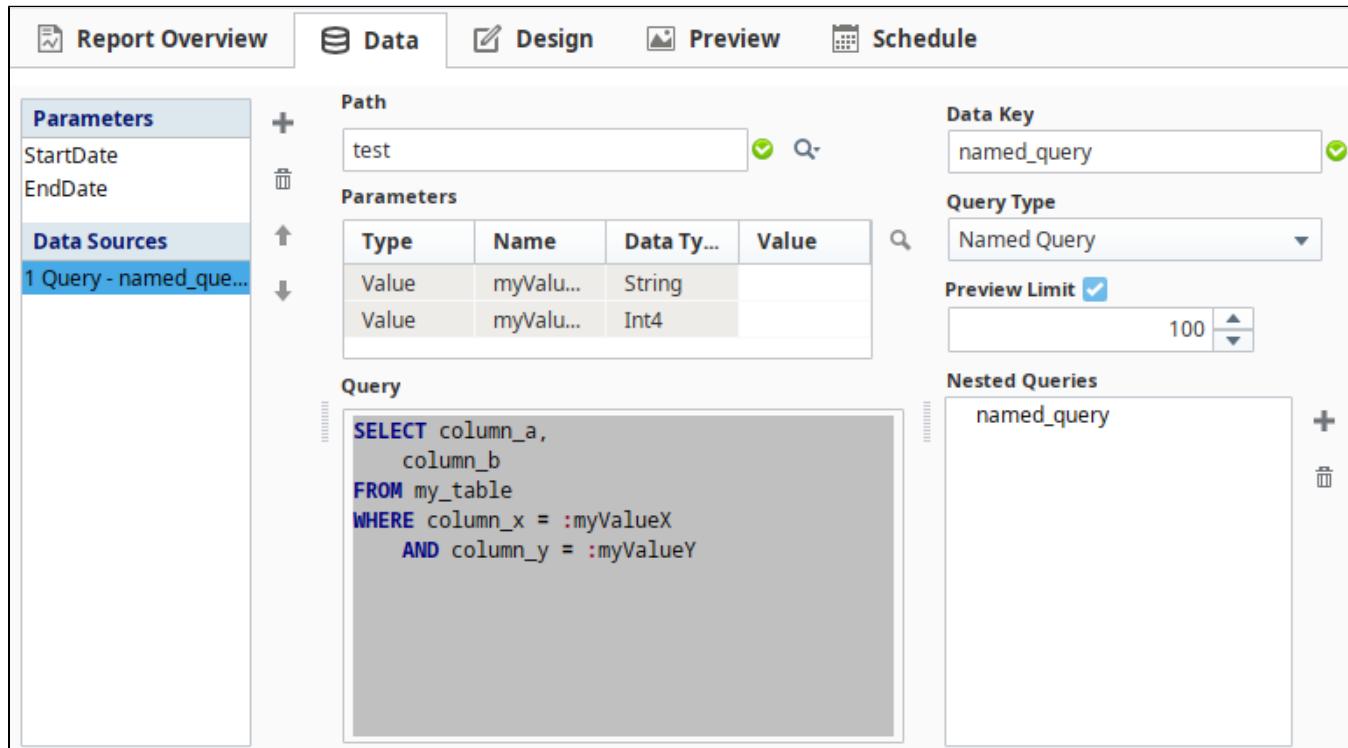
A [Query Binding](#) type has been added to leverage named queries for any component property bindings. [Named Query Parameters](#) may be bound to Tags or other properties on the same window, allowing your users to modify the resulting dataset by manipulating other components similar to the

original SQL Query binding type. You can also select and update a query to simulate a bi-directional binding to the database.



Reporting

A [Named Query Data Source](#) has been added to Reports. Report Parameters may be used by the Named Query to generate dynamic reports.



Scripting

The [system.db.runNamedQuery](#) function may be used to called a Named Query from any Python script. This provides a secure method to execute a query from any location in Ignition.

Related Topics ...

- [system.db.runNamedQuery](#)
- [Named Query Data Source](#)
- [Named Query Bindings](#)

In This Section ...

Named Query Workspace

Named Query Workspace Overview

Named Queries have a dedicated workspace inside of the Project Section of the Designer. This workspace allows for the creation and testing of Named Queries. Once created, the Named Query may be called from another resource, such as a [datasource in a report](#) or a component using a [Named Query Bindings](#).

Named Queries are created by right-clicking the Named Query item in the Project Browser. Like other resources in the Project Browser, Named Queries can be organized in folders, which creates a unique path to the query and helps keep your queries organized.

Note: Named Queries are referenced by path, so renaming the Named Query or any parent folders will require you to update the path on any other resources that are using it.

Also like other resources, multiple Named Queries may be opened in the same Designer session. Tabs at the bottom of the Designer allow for easy swapping between Named Queries.

On this page ...

- [Named Query Workspace Overview](#)
- [Workspace Sections](#)
 - [Settings](#)
 - [Authoring](#)
 - [Testing](#)

Workspace Sections

The Named Query workspace contains three tabs: Settings, Authoring, and Testing. A description of each section follows.

Settings

The Settings  tab contains configuration properties and security for the selected Named Query.

Test Query

 Settings

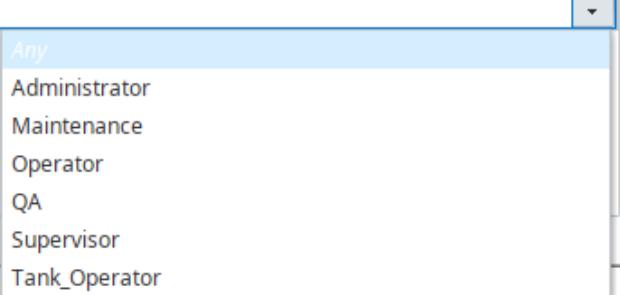
 Authoring |  Testing

Named queries allow a higher level of security for database queries and allow queries to be written once, tested, and easily reused throughout the project.

Enabled

Security

Current zone and role must match one of these entries to execute this query. See 'Security' under the Configure tab in the Gateway web page to learn more.

Security Zone	Role
Ridgefield East	 <ul style="list-style-type: none">AnyAdministratorMaintenanceOperatorQASupervisorTank_Operator   
Description	
Caching	<input type="checkbox"/> Cache Query Results  1 Seconds
Test Query 	

The following properties are available:

Item Name	Description
Enabled	Determines if the Named Query is enabled or disabled. A disabled Named Query may not be executed.
Security	Specifies a combination of Security Zones and Roles that may call the Named Query. Only roles in the projects user source will be available in the Role dropdown. Multiple rows may be configured to account for granular access restrictions (i.e., requests from Administrator roles originating from the Office security zone area could be allowed, while requests from same users in the plant floor zone could be denied). If the request does not match any of the specified zone and role combinations, then the query will not run. Additionally, if a ScalarQuery type has a Fallback value configured, that Fallback value will not be returned either: the query will not execute due to security settings, so there is never a chance for other errors to occur. Either the Security Zone or the Role (but not both) may be left blank. This means it is available to all objects of that type (i.e., with a blank Role and the "office" Security Zone, all roles in the "office" zone are valid).
Description	Allows you to give the Named Query a description.
Caching	Allows the Gateway to cache the results of the query. See the Named Query Caching page for more details.

Authoring

The Authoring  tab is where the query and parameters are created. There is also a Table Browser and [Query Builder](#) that can be used to help you to create your query.

Test Query

Settings Authoring Testing

Database Connection **Query Type**

<Parameter> Query

Parameters

Type	Name	Data Type
Value	myValueX	String
Value	myValueY	Int4

Table Browser

Query

```

1 SELECT column_a,
2     column_b
3 FROM my_table
4 WHERE column_x = :myValueX
5     AND column_y = :myValueY

```

Query Builder

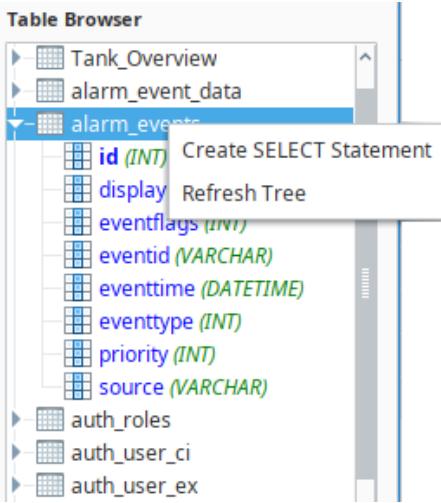
Builder Syntax

Universal

Test Query ×

Item Name	Description																
Database Connection	<p>The database connection the Named Query should run against. In addition to a list of the database connections configured in the Gateway, this dropdown contains two unique values: <Default> and <Parameter>.</p> <ul style="list-style-type: none"> • <Default>: The query will execute against the project's default database connection. • <Parameter>: The query expects the database connection name to be passed in as a parameter when called. This allows you to use a dynamic database connection. <p>Note:</p> <p>A parameter for the database connection does not need to be manually created in the Parameters section. Instead, the Named Query will have a special "Database" type parameter available when called. Below is an image of a Named Query binding that is utilizing the <Parameter> connection type.</p> <p>SELECT Query</p> <p>Path</p> <p>Test Query</p> <p>Parameters</p> <table border="1"> <thead> <tr> <th>Type</th> <th>Name</th> <th>Data Type</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>Database</td> <td>database</td> <td>String</td> <td></td> </tr> <tr> <td>Value</td> <td>myValueX</td> <td>String</td> <td></td> </tr> <tr> <td>Value</td> <td>myValueY</td> <td>Int4</td> <td></td> </tr> </tbody> </table>	Type	Name	Data Type	Value	Database	database	String		Value	myValueX	String		Value	myValueY	Int4	
Type	Name	Data Type	Value														
Database	database	String															
Value	myValueX	String															
Value	myValueY	Int4															

Query Type	<p>SELECT Query</p> <p>Path</p> <p>Test Query ✓ 🔍</p> <p>Parameters</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Type</th><th>Name</th><th>Data Type</th><th>Value</th></tr> </thead> <tbody> <tr> <td>Database</td><td>database</td><td>String</td><td></td></tr> <tr> <td>Value</td><td>myValueX</td><td>String</td><td></td></tr> <tr> <td>Value</td><td>myValueY</td><td>Text</td><td></td></tr> </tbody> </table> <p>The type of query to execute. The following options are available:</p> <ul style="list-style-type: none"> Query: Allows SELECT Queries and returns a full dataset. This type should be selected when running a SELECT statement that returns multiple rows or columns. ScalarQuery: Allows SELECT queries and returns a single value. This type should be used when running SELECT statements that only return a single value. The very first cell returned will be the only output. This type is special in that a Fallback value may be defined. The Fallback value will be returned if the Named Query would return an error. Note, the Fallback parameter will not be returned if the request does not meet the security requirements. UpdateQuery: Allows all UPDATE types of queries (queries that mutate or otherwise modify rows on a table) and returns the number of rows affected by the query. This type should be selected when modifying the database in some way, such as when running an INSERT, UPDATE, or DELETE query. 	Type	Name	Data Type	Value	Database	database	String		Value	myValueX	String		Value	myValueY	Text	
Type	Name	Data Type	Value														
Database	database	String															
Value	myValueX	String															
Value	myValueY	Text															
Parameters	<p>A table of the parameter names and types that will be used in the query. These parameters have three types, Value, QueryString, and Database. Most commonly, the Value type is used and can be accessed by using the :paramName notation. More details on this field may be found on the Named Query Parameters page.</p>																
Query	<p>The query that will execute when the Named Query is called. You can type directly into this field or use the Table Browser on the right to get started. Right-Clicking inside this field will cause a popup menu to appear:</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> <p>Query</p> <pre>1 SELECT column_a, 2 column_b 3 FROM my_table 4 WHERE column_x = :myValueX 5 AND column_y = :myValueY</pre> </div> <div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;"> <p>Parameterize ➔</p> <p>Insert Parameter ➔</p> <p>Undo</p> <p>Can't Redo</p> <hr/> <p>Cut</p> <p>Copy</p> <p>Paste</p> <hr/> <p>Delete</p> </div> <div style="border: 1px solid #ccc; padding: 5px; background-color: #f9f9f9;"> <p>Select All</p> <hr/> <p>Themes ➔</p> </div> <p>Most of the items on this menu are self-explanatory, but a few require special mention:</p> <p>Parameterize: Contains two sub-menu items, which are detailed below. Note, that the sub-items will be disabled unless you right-click on some text that does not reference a parameter, table name, or column name in the query.</p> <ul style="list-style-type: none"> Make Value: Turns the selected text into a value-type parameter. The new value-type parameter will appear in the Parameter table above the query. Make QueryString: Turns the selected text into a QueryString. The new QueryString will appear in the Parameter table above the query. Note that QueryStrings are susceptible to SQL injection attacks. Because of this, the Make Value option is recommended over this option. <p>Insert Parameter: Quickly creates a reference to the selected parameter. This menu is an alternative to dragging-and-dropping from the Parameters table or typing the name of the parameter.</p>																

Table Browser	<p>Provides a list of the tables in the selected Database Connection. Tables may be dragged into the Query field to quickly insert the name of the table. Additionally, right-clicking on a table in the list will cause a popup menu to appear:</p>  <ul style="list-style-type: none"> • Create SELECT Statement will populate the Query field with a SELECT statement targeting the selected table or selected row of the table. • Refresh Tree will refresh the Table Browser.
Query Builder	Opens the Query Builder , which provides an easy way to create SQL queries using a drag-and-drop interface. This button will be disabled if the Database Connection property is set to <Parameter>. This is the same Query Builder used in other places like the reporting data page.
Builder Syntax	Specifies the syntax the Query Builder should use. Contains syntax for many popular databases, and has a Universal selection that should work in most scenarios.

Testing

In the Testing  tab, you can test your query without leaving the workspace. Fill in your values and click the **Execute Query** button to see your results.

Test Query

[Settings](#) | [Authoring](#) [Testing](#)

Test Parameters

Type	Name	Data Type	Value
Database	database	String	
Value	myValueX	String	
Value	myValueY	Int4	

Use Sample Size Limit

100

[Execute Query](#)

[Export to CSV](#)

Results

result

1 row fetched in 0.071s

Test Query [X](#)

Item Name	Description
Test Parameters	Allows you to manually supply test values to the parameters to the queries. The table is populated by the Parameters field on the Authoring section.
Use Sample Size Limit	When checked, allows you to set the maximum number of rows the query will return while testing the Named Query. In addition this setting also limits the results of that named query called elsewhere in the Designer. This property is only enabled when the Query Type on the Authoring section is set to Query .
Execute Query	Runs the Named Query using the parameter values listed above. The Results area will display any results returned by the query.
Export to CSV	Exports the results of the query to a CSV file. The button becomes available after results are returned. Note that this will only return the values shown, check the Sample Size Limit when using this button.
Results	The results returned by the Named Query when testing. Populated by the Execute Query button.

Related Topics ...

- [Named Query Parameters](#)
- [Named Query Conversions](#)
- [Named Query Caching](#)

Named Query Parameters

Parameters allow you to make Named Queries dynamic. They act as placeholders you can pass values into when requesting the query to execute. Other resources in Ignition can then pass arguments into the parameters. The exact implementation depends on what resource is requesting the Named Query, such as a [Named Query Bindings](#), a [Named Query Data Source](#), or the `system.db.runNamedQuery` function. Check out the [Named Query Example](#) page for a complete example on passing parameters into a named query.

Types of Named Query Parameters

There are three types of Parameters in Named queries. Each varies in usage.

Value

The Value type should be used whenever a Named Query needs a dynamic WHERE clause. These act like values passed to a prepared statement, meaning they can never be used to parameterize column or table names. However, they are resilient to SQL injection attacks.

SQL Query - Using a Parameter

```
SELECT * FROM mytable  
WHERE name = :myParam
```

On this page ...

- [Types of Named Query Parameters](#)
 - [Value](#)
 - [QueryString](#)
 - [Database](#)
- [Parameters while Authoring a Named Query](#)
 - [Creating Parameters](#)
 - [Using Parameters in the Query](#)

QueryString

QueryStrings are more flexible than the Value type in that they can be used to parameterize column and table names. However, their values are never sanitized, which causes them to be more susceptible to SQL injection attacks. When using QueryStrings, it is best to avoid situations where the user can manually type in the value that will be passed to the Named Query. Additionally, if you are using a QueryString for a string in the where clause, you would need to provide quotation marks.

SQL Query - A Using QueryString

```
SELECT {myColumnName} FROM mytable  
WHERE name = '{myName}'
```

Database

Database type parameters cannot be created manually. Instead, it is automatically created when the **Database Connection** dropdown on the **Authoring** section is set to **<Parameter>**. Additionally, this parameter is not used in the body of the query. This type allows you to parameterize the database connection when the Named Query is called. This way the Named Query can run against multiple database connections specified by the resource that made the request.

Parameters while Authoring a Named Query

Creating Parameters

New parameters can be created in the Authoring section of a Named Query by clicking the Add  icon next to the Parameters table.

test

Settings Authoring Testing

Database Connection Query Type

<Default> Query

Parameters

Type	Name	Data Type
Value	myValueX	String
Value	myValueY	Int4
QueryString		
Value		

Query

```
1 | SELECT column_a,
```

i In Named Queries, Parameters are referenced by their name, so renaming the Parameter will require you to update it on any other resources that are using it.

Each parameter has three properties represented by different columns in the table. These may be edited by double-clicking on the cell you wish to modify:

- **Type:** Changes the type between Value and QueryString.
- **Name:** Determines the name of the parameter, and how it will appear in the query. Names are **not** case-sensitive and must be unique. Additionally, they may only use letters, numbers, dashes and underscores.
- **Data Type:** Specifies the datatype of the parameter. The Type of the parameter determines which data types are available. Note, that QueryStrings may only be configured as strings, where as Value-type parameters have more types available.

Using Parameters in the Query

Once created, parameters can be inserted into the Query field by **drag-and-drop** from the Parameter table onto the Query field, or by using the **right-click menu** in the Query field.

Query

```
1 | SELECT column_a,
2 |     column_b
3 | FROM my_table
4 | WHERE column_x = :myValueX
5 |     AND column_y =
```

Parameterize ▶

Insert Parameter ▶

myValueX
myValueY

Undo

Can't Redo

Additionally, the parameters may be typed in manually, but the correct syntax must be used.

Related Topics ...

- [Named Query Bindings](#)
- [system.db.runNamedQuery](#)
- [Named Query Conversions](#)
- [Using Named Queries - Example](#)

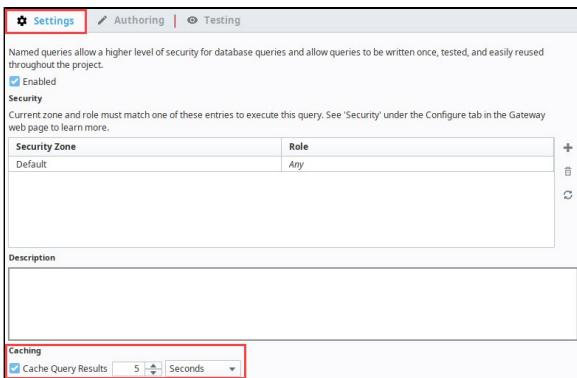
Named Query Caching

Overview

Named queries can opt-in to caching the results on the Gateway. This means if another request to the same Named Query comes in, the Gateway can return the cached result instead of having the database executing the query again. This will use more memory on the Gateway (to maintain the results), but could result in less queries running against the database.

Named Query caching is disabled by default, but can be enabled on the **Settings** section of each Named Query. If caching is enabled, the spinner and dropdown fields set the **lifespan** of the cache. Once the lifespan expires, the cache is invalidated. After the cache is invalidated, the next request for the Named Query will force the Gateway to re-execute the query and build a new cache.

Caching is especially useful for tables that are not updated often like recipe or inventory tables. Tables that update often like historical storage tables are bad candidates.

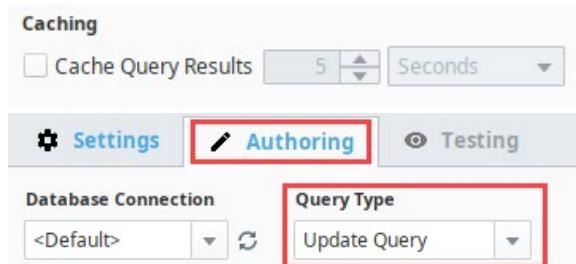


On this page ...

- Overview
 - When Caching is Disabled
- Scenario
- Considerations
 - Gateway Memory
 - How often the Database Values are Updated
 - How often the Named Query is Executed
- Cached Query Updates and Designer Values
- Scripting Functions to Clear Cache from a Named Query

When Caching is Disabled

Update queries are not allowed to cache their results. With UPDATE or DELETE statements, the work will already have been performed after the first execution, and INSERT statements typically utilize different parameters each execution. Thus if the Caching setting is disabled, then this typically means that the **Query Type** in the **Authoring** tab is set to an **Update Query**.



Scenario

Consider the following:

- A Named Query is created. This runs a query to fetch data from an arbitrary table.
- A component in a project is configured to call the Named Query on a window.
- **Client_A** navigates to the window, which triggers a request for the Named Query be executed.
- Several seconds later, **Client_B** opens the same window, and needs the same results.

In this scenario, if caching was enabled on the Named Query, then **Client_B** would not cause another query execution (assuming both **A** and **B** passed the same values to the parameters). This would result in less network traffic between the Gateway and the Database, and less work for the Database. An example with two clients isn't exciting, but the same scenario with ~50 clients would mean a huge potential performance boost, especially if those requests were polling at a 5 second rate, and the cache period was configured to 5 seconds.

Considerations

The following are considerations that should determine whether or not caching the results of a Named Query is helpful.

Gateway Memory

Each time the Named Query is called, if the arguments passed differ from those used to create the current cache, a new cache is created. This means Named Queries that are frequently called with varying parameter values will create multiple caches. If the results are large datasets, this can result in a large amount of the Gateway Memory being tied up maintaining these caches. In this scenario, you will want to monitor the memory usage of the Gateway.

This is especially important to consider when dealing with queries that accept a timestamp parameter that uses an expression like now() that will return time to the current second. If 5 clients are opened with each one 1 second apart, they will all have different timestamps and create separate cache entries.

How often the Database Values are Updated

Once a cache is created, the Named Query will not look for any changes made to the database table until the query next executes. This means changes, such as adding a new row to the table, will not appear in the Named Query's results until the cache is invalidated. Data that changes often may not be a good candidate for caching. In this case, the lifespan of the cache should be set to a short amount of time, depending on how often the data may get updated.

How often the Named Query is Executed

If a Named Query is called frequently and there are not many updates to the table data, then there can be a huge performance benefit to caching the results. Fewer calls to the database result in less network traffic and better overall performance.

Cached Query Updates and Designer Values

Cached query results have some notable interactions in the Designer. Specifically:

- **Named Queries results executed in the Designer never cache:** This is because the Designer uses the live version of the query in the Designer instead of the saved Gateway version. This also prevents bad results being stored in the cache, and then appearing in the Clients.
- **Making changes to a Named Query and saving will invalidate all caches for that Named Query:** When a change is made to a Named Query and saved, the new query is pushed to the Gateway, which means all current caches are immediately outdated.

Scripting Functions to Clear Cache from a Named Query

You can also clear the Named Query cache using the following scripting functions.

- [system.db.clearNamedQueryCache](#) - Clears the cache of a single Named Query.
- [system.db.clearAllNamedQueryCaches](#) - Clears the caches of all Named Queries.

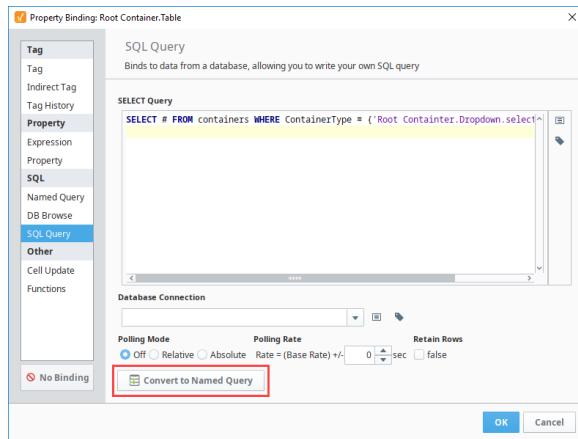
Related Topics ...

- [Named Query Conversions](#)
- [Using Named Queries - Example](#)

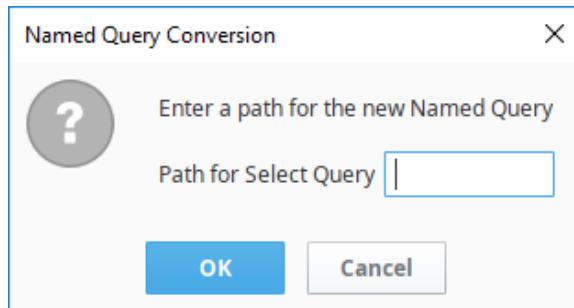
Named Query Conversions

Converting SQL Queries to Named Queries

If you have a SQL Query binding on a component, it is easy to convert it to a Named Query. Simply open up the binding and click the **Convert to Named Query** button.



A popup appears where you can enter a path to the new Named Query.



On this page ...

- [Converting SQL Queries to Named Queries](#)
- [Converting with an Update Query](#)
- [Modifying Converted Query to Use Parameters](#)



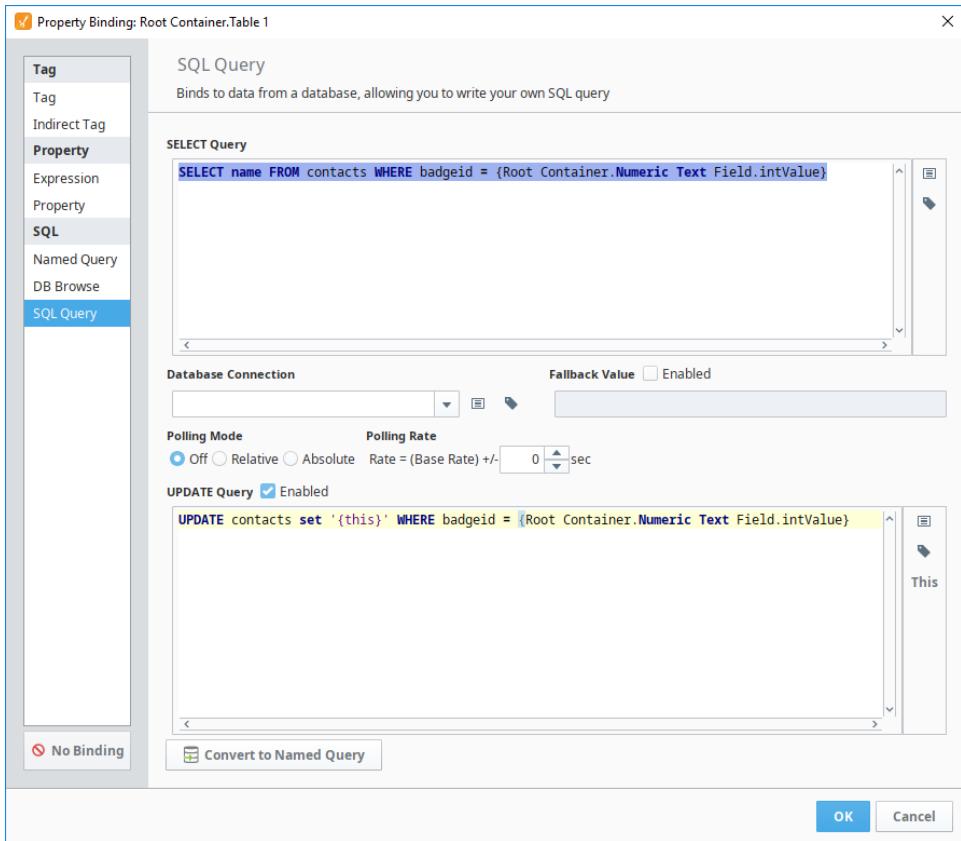
Named Query Conversion Example

[Watch the Video](#)

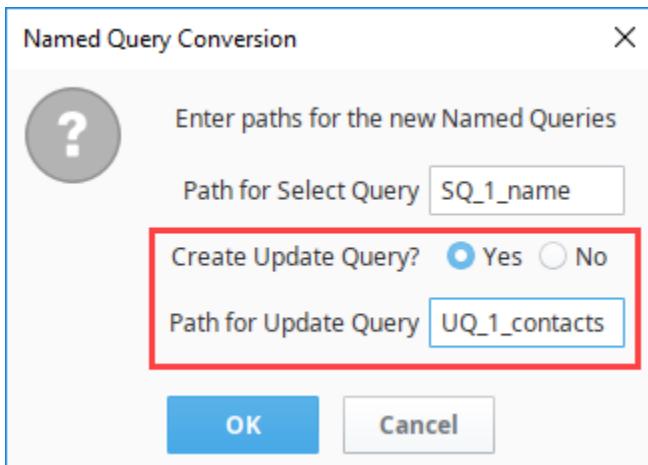
Converting with an Update Query

If your SQL Query binding is on an component property whose value can be updated and written back to the database (for example, input components and button components), you have the option to create two Named Queries: one for the Select Query in the binding and a second for the Update Query.

1. Enter the Select Query.
2. Click the Enabled checkbox for the Update Query.
3. Enter the Update Query.
4. Click OK.



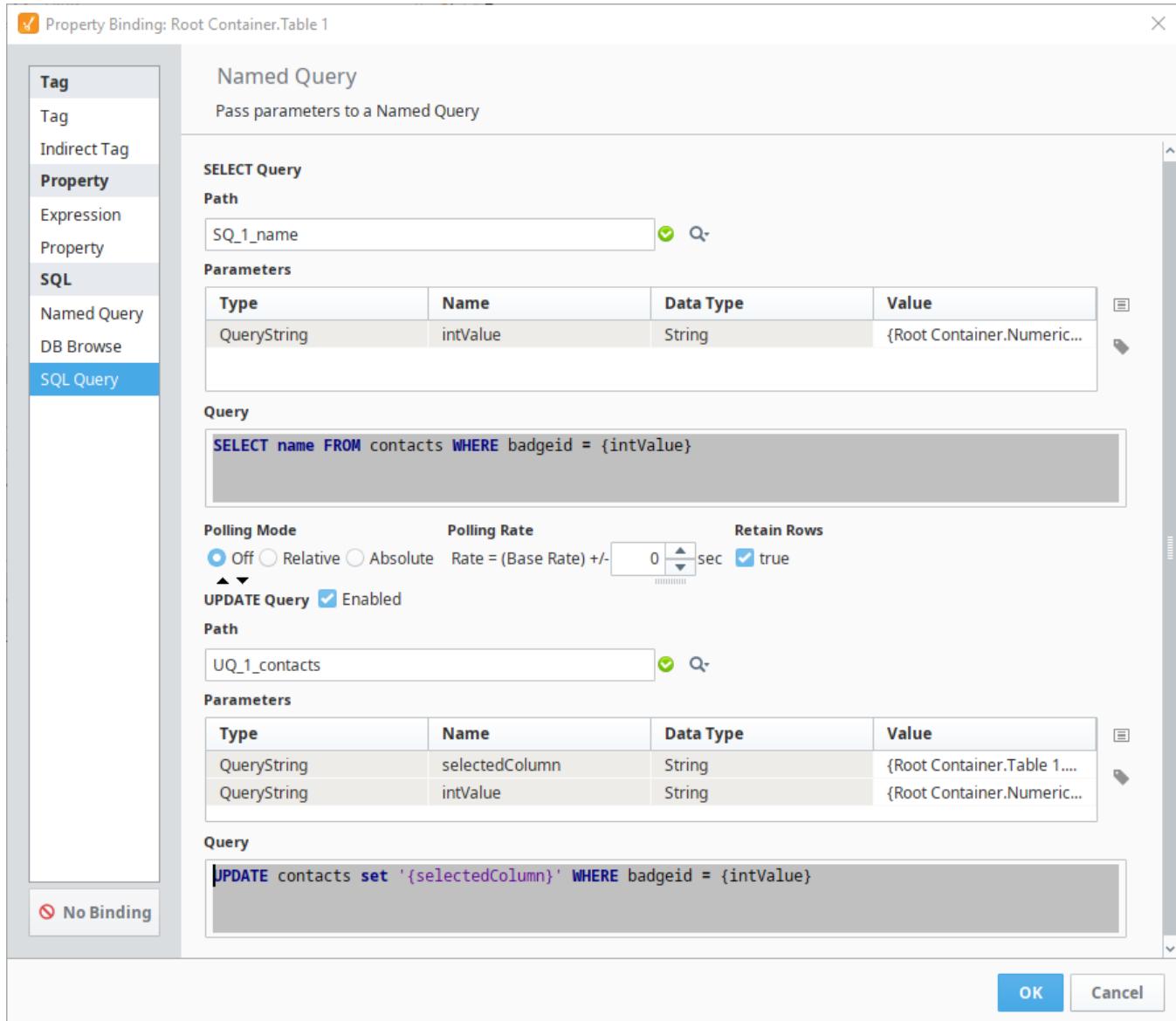
5. In the popup, enter the path for the Select Query.
6. Click the Yes button next to Create Update Query,
7. Enter the path for the Update Query,
8. Click Save.



Warning for 7.9.5 and Prior

When converting a SQL Query Binding that contains an UPDATE Query in 7.9.5 and prior, an Update query will not be generated. Thus, it was possible to lose the Update Query upon conversion.
If using these versions of Ignition, it is highly recommended to manually create the Update Named Query before pressing the Conversion button. As of 7.9.6, this is no longer an issue, and the **Convert to Named Query** button may freely be used.

After confirming the conversion, the Window will show the query/queries. Click the **OK** button to save this change.



When converting in this manner, all parameters will be created at **QueryStrings**. It is highly recommended that you modify your new Named Queries so that these values become **Value Parameters** instead, as mentioned later on this page.

Modifying Converted Query to Use Parameters

When converting a query to a Named Query, it is **strongly** recommended that you go back into the query and convert the parameter from a **QueryString** type to a **Parameter** type. To do this, you first need to change all **QueryString** type parameters to the **Parameter** type. This is as simple as selecting **Parameter** from the dropdown under **Type**.

Now that your parameter(s) types have been modified, the second thing you need to do is modify syntax in the query to use the parameter(s). The **Parameter** and **QueryString** types are referenced differently in the query, so you will need to ensure that you modify how the parameter is referenced. In addition because the **Parameter** type works like a prepared statement, it does not need any quotation marks around any string type parameters like a **QueryString** would, so all quotation marks around parameters should be removed from the query. See the images below for an example.

From this:

To this:

Once the syntax in the query has been modified, test it out in the **Testing** section at the top to make sure everything works correctly. If so, then you have successfully converted to a Named Query.

Related Topics ...

- [Named Queries](#)
- [Named Query Bindings](#)
- [SQL Query Bindings in Vision](#)

Using Named Queries - Example

A Named Query Example

Here we will go over the steps necessary to put together a basic named query. This example will create a Named Query that uses a single parameter to run a select query, and then add a Table to a window and create a binding that uses our new Named Query.

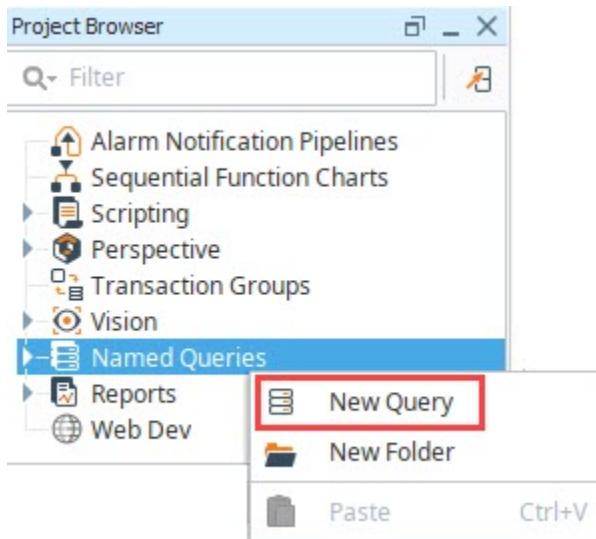
We are going to be querying a table in the database that holds information about products stored in storage bays. The database table is named "containers" and has the following structure.

id	StorageBay	ContainerType	ItemName	Weight	Time
1	1	Jug	Vanilla	25.2	2017-06-25 15:58:47
6	1	Mason Jar	Chocolate	12.3	2017-06-26 16:05:27
18	2	Tray	Swiss Cheese	88.8	2017-06-25 01:21:31
22	3	Tray	Cheddar Cheese	54.7	2017-06-25 03:52:16
23	3	Basket	Strawberry	36.8	2017-06-25 15:56:03
31	3	Jug	Whole Milk	80.1	2017-06-27 09:51:31
32	3	Jug	Fat Free Milk	76.9	2017-06-27 09:52:52

If you want to follow along with the example, feel free to make a database table that looks similar to this one and add as many rows of data as you would like, otherwise, you can use your own and substitute in the proper column names from your database.

Creating a Named Query and Adding Security

1. Start by opening up the Designer and [loading a project](#).
2. Locate the Named Query section of the project browser, right click on it, and select the **New Query** option.



3. You should now have a fresh Named Query that you can rename whatever you want. We used **FirstNamedQuery** in the example.

On this page ...

- [A Named Query Example](#)
 - [Creating a Named Query and Adding Security](#)
 - [Building the Query](#)
 - [Using the Query](#)

The screenshot shows the Project Browser interface with the 'Named Queries' section expanded. The 'FirstNamedQuery' item is selected. The 'Settings' tab is active, showing the following configuration:

- Enabled:** Checked
- Security:** Current zone and role must match one of these entries to execute this query. The table shows:

Security Zone	Role
Any	Any
- Description:** (Empty)
- Caching:** Cache Query Results: 1 Seconds

- To setup security on the Named Query, set required Security Zone and Role combinations in the Security table of the Settings tab. You can leave this blank if you don't have **roles** or **zones** set up yet.
- If multiple security combinations are required, use the Add icon to add additional rows.

The screenshot shows the 'FirstNamedQuery' settings page. The 'Security' table is displayed, showing security zones and their associated roles. A red box highlights the dropdown menu for selecting a role, which is currently set to 'Maintenance'. Other options in the menu include 'Any', 'Administrator', 'Operator', 'QA', 'Supervisor', and 'Tank_Operator'.

Security Zone	Role
Any	Administrator
Production West	Maintenance
	Any
	Administrator
	Operator
	QA
	Supervisor
	Tank_Operator

Building the Query

- Click on the **Authoring** tab. Here is where we do most of the work.
- Under **Database Connection**, we need to select a database connection that this named query will use. We selected **<Default>**.
- For the **Query Type**, we can decide what type of query this will be. For this example, we are running a select query that will return a dataset, so we chose **Query**.

4. In the Parameters section, we can decide on a list of parameters that will be used in this query. This query is fairly simple and will only use a single parameter.

Click the Add  icon to add a new parameter and set the following values:

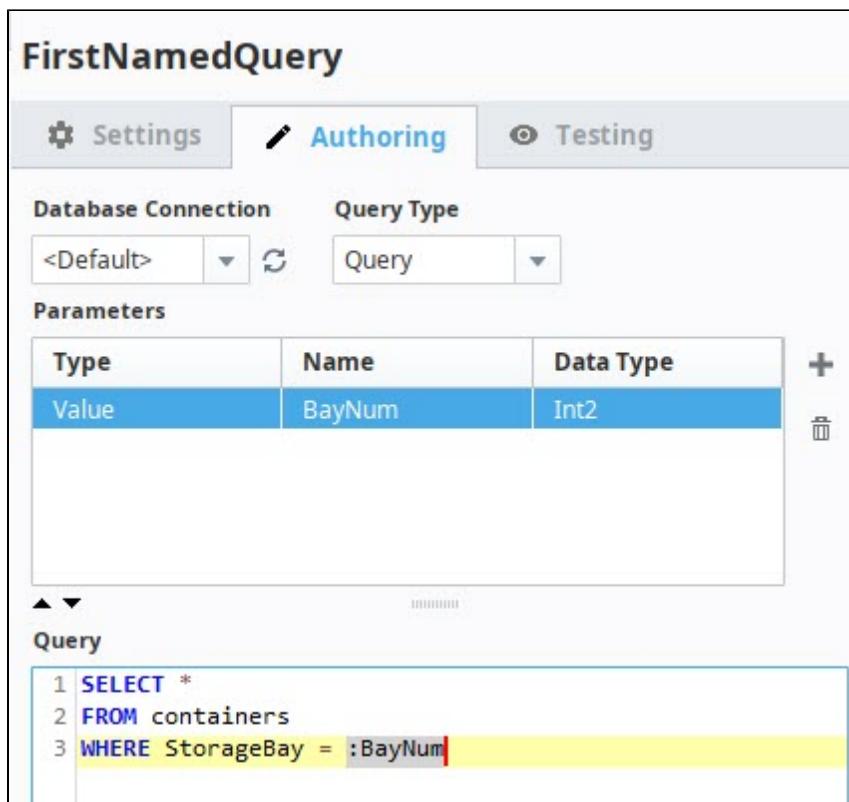
Type: Parameter

Name: BayNum

Data Type: Integer

5. The Query section below is where we construct our query using the Table Browser.

- Right click on the containers table in the Table Browser on the right, and click on **Create SELECT Statement**. This will populate our query field with a basic select all statement.
- Type into the **Query** field and add the following WHERE clause: "**WHERE StorageBay =**"
- Now drag the **BayNum** parameter from the Parameters table to the end of the query you just typed. Notice "**:BayNum**" will be added at the end of the query.



The screenshot shows the 'FirstNamedQuery' configuration window. The 'Authoring' tab is selected. At the top, there are 'Database Connection' and 'Query Type' dropdowns. Below them is a 'Parameters' table with one row: Value (Type) and BayNum (Name). The 'Data Type' column shows Int2. To the right of the table are a '+' button for adding more parameters and a '-' button for deleting existing ones. At the bottom, the 'Query' section contains the following SQL code:

```

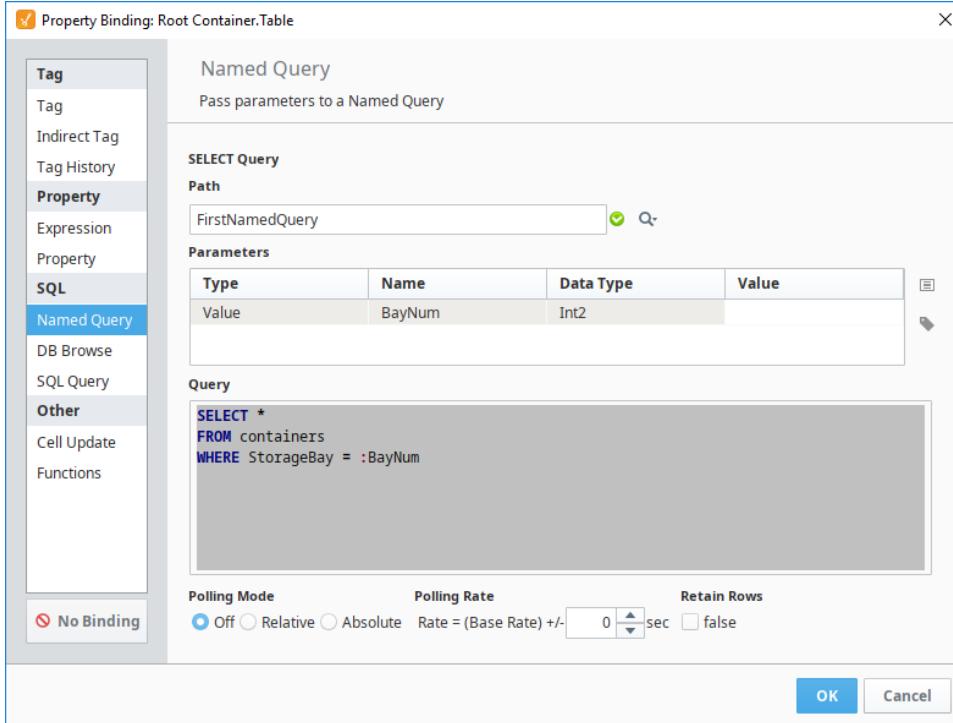
1 | SELECT *
2 | FROM containers
3 | WHERE StorageBay = :BayNum

```

Using the Query

- In the Project Browser, create or open a [Main Window](#).
- Drag a **Dropdown List** component onto the window.
- The Dropdown List is where we will be able to select a Bay Number to use as our Named Query's BayNum Parameter. Use the [Dataset Viewer](#) to set the **Data** property of the Dropdown to look like this:

Value	Label
1	Bay 1
2	Bay 2
3	Bay 3



4. Now drag a **Table** component onto the window. We can setup a Named Query binding on the Data property.
5. Click on the **Binding** icon for the **data** property and select the **Named Query** binding type.
6. Set the Path property by clicking on the **Select Resource Path** icon and selecting your new Named Query from the list. For this example, it is **FirstNamedQuery**. Alternately, you can type the name in.
7. Highlight your **BayNum** Parameter (in the Parameters table) and click on the Insert Property icon. We want to select the **Selected Value** property of our Dropdown List.
8. Finally, we want to ensure the Polling Mode is set to **Off**. This means the query will not run continuously, but will only run when it changes such as when a new bay is selected from the dropdown.
9. Click the **OK** button to save your binding, and put your Designer into **Preview Mode** to test it. We can then make a selection (i.e., Bay 3) with our Dropdown, and see the table populate with data.

id	StorageBay	ContainerType	ItemName	Weight	Time
22	3	Tray	Cheddar Cheese	54.7	Jun 25, 2017 3:5..
23	3	Basket	Strawberry	36.8	Jun 25, 2017 3:5..
31	3	Jug	Whole Milk	80.1	Jun 27, 2017 9:5..
32	3	Jug	Fat Free Milk	76.9	Jun 27, 2017 9:5..

Bay 3

Related Topics ...

- [Named Queries](#)

Queries in Scripting

Overview

In addition to using a binding, queries can be called from a Python script. This typically involves using one of Ignition's built-in [Scripting functions](#). This page presents several approaches to interacting with a database from a Python Script. See the [Scripting](#) section for more information about using Python.

Named Queries

Named Queries may be called from a Python script using the `system.db.runNamedQuery` function. Named Queries can execute as a Prepared Statement when not utilizing [Query String parameters](#), so they can offer a secure approach to calling a query from a script.

Additionally, Named Queries are incredibly easy to call from a script, since you only need to know their name, and don't have to bother with SQL syntax in the script. This is another way to protect your database from other users in the Designer, as you can [Protect Named Queries](#).

Python - Calling a Named Query

```
namedQuery = "Add New Order"
parameters = { "accountId":123, "productName": "Bananas" }

system.db.runNamedQuery(namedQuery, parameters)
```

For more information, refer to the [Named Queries](#) section.

On this page ...

- [Overview](#)
- [Named Queries](#)
- [Prepared Statements vs Standard Statements](#)
 - [Standard Statements vs Prepared Statements](#)
 - [How do I Call a Prepared Statement?](#)
 - [Standard Statements](#)
- [Stored Procedures](#)
- [Transactions](#)

Prepared Statements vs Standard Statements

A prepared statement is a feature in a database that executes a parameterized query. One of the main reasons to utilize a prepared statement is because they are resilient to SQL Injection Attacks. Because of this, we **highly recommend** you utilize Prepared Statements over Standard Statements, but Named Queries are the most secure. Especially in cases where the users can type values that will be included in the query.

Prepared Statements typically involve passing two pieces of information to the database:

- A query string that contains placeholders to represent where arguments will be passed in later. These are represented in the query as question mark characters ("?")
- A series of arguments to replace the placeholder characters in the static query.

Standard Statements vs Prepared Statements

Typical SQL insert queries look like the following:

SQL - Standard Statement

```
INSERT INTO orders (account_id, product_name) VALUES (123, 'Bananas')
```

A Prepared Statement instead looks like the following. Note, that the placeholders do not require quotation marks even when a string will be passed in.

SQL - Prepared Statement

```
INSERT INTO orders (account_id, product_name) VALUES (?, ?)
```

How do I Call a Prepared Statement?

Prepared Statements can be called from a script using specific functions. Typically, they contain "Prep" in the name such as `system.db.runPrepQuery`, or `system.db.runPrepUpdate`. When in doubt, take a look at the sub pages in the [system.db](#) section.

There are typically two required parameters with these functions: a string literal that represents the query, and a list of parameters.

Python - Calling an Insert Query as a Prepared Statement

```
query = "INSERT INTO orders (account_id, product_name) VALUES (?, ?)"  
args = [123, "Bananas"]  
system.db.runPrepUpdate(query, args)
```

Notable Prepared Statement Functions

- [system.db.runPrepQuery](#): Should be used when SELECT statements are used, as it returns a dataset containing the results. This function is effectively "read-only", as it does not manipulate data in the database.
- [system.db.runPrepUpdate](#): Should be used when manipulating the database in some way. When executing a statement that utilize INSERT, UPDATE, or DELETE, the runPrepUpdate function should be called. Note that it does return the number of rows affected, so the return value can be used to keep track of how much of an impact the query had.
- [system.db.runScalarPrepQuery](#): Like runPrepQuery above, but only returns the first column of the first row: i.e. a single value is returned instead of a full dataset. This is useful when using an aggregate function of some sort to return a count or total, as it saves your script the work of extracting the value from the full dataset that runPrepQuery normally returns.

Standard Statements

Queries can be called as a Standard Statement (a statement that that isn't a Prepared Statement) by using the [system.db.runQuery](#) and [system.db.runUpdateQuery](#) functions. However, these are susceptible to [SQL Injection attacks](#), and should be avoided where possible: especially when users have access to a keyboard and can directly type values that will be used in the query.

Calling a Standard Statement involves building the entire query as a single string, and passing the string on to our Standard Statement functions.

Python - Calling an Insert Query as a Standard Statement

```
query = "INSERT INTO orders (account_id, product_name) VALUES (%i, '%s')" % (123, "Bananas")  
system.db.run(query)
```

Notable Standard Statement Functions

- [system.db.runQuery](#): Executes a SELECT statement, returning a result set as a dataset.
- [system.db.runUpdateQuery](#): Executes a statement that manipulates the database in someway. Should be used with INSERT, UPDATE, and DELETE statements.
- [system.db.runScalarPrepQuery](#): Similar to runQuery, except only a single value is returned: the first column of the first row. Generally used in conjunction with SELECT statements that contain an aggregate function.

Stored Procedures

If your database administrator has already configured Stored Procedures for you to use, then they can easily be called from a Python Script. Using Stored Procedures in a script typically involves two main steps:

1. A SProcCall object is created with the [system.db.createSProcCall](#) function. The SProcCall object contains several functions that can be used to register parameters, and access the results set returned by the Stored Procedure after it has been executed.
2. The [system.db.execSProcCall](#) function must be used to execute the Stored Procedure.

Python - Creating and Executing a Stored Procedure

```
# Create a SProcCall object, which will be used to configure parameters on the Stored Procedure, and then  
# executed.  
myCall = system.db.createSProcCall("insert_new_order")  
  
# Register parameters on the SProcCall object.  
myCall.registerInParam(1, system.db.INTEGER, 123)  
myCall.registerInParam(2, system.db.VARCHAR, "Bananas")  
  
# Execute the Stored Procedure.  
system.db.execSProcCall(myCall)
```

Take a look at the [SQL Stored Procedures](#) page for more details.

Transactions

A SQL Transaction can also be executed from a script. For the unfamiliar, a Transaction is a batch of statements that will be executed together, and either succeed or fail as a group. Note, that the statements executed in the Transaction are not visible by other connections in the database until you commit them.

Transactions typically involve several steps:

1. Call `system.db.beginTransaction`. This returns a **transaction identifier** that can be used with other statements. Using this identifier is how you specify that a statement should be included in the transaction.
2. Start calling other statements with other functions, such as `system.db.runPrepUpdate`. The function's "tx" parameter will be passed the **transaction identifier**.
3. Commit or Rollback the transaction. Use `system.db.commitTransaction` to commit, and `system.db.rollbackTransaction` to rollback. These options are essentially the same as applying or canceling the results of the queries. Committing will make the updated results available to other database connections.
4. Close the Transaction once you're done, which can be accomplished with the `system.db.closeTransaction` function. This invalidates the **transaction identifier**.

Python - The SQL Transaction Workflow

```
# 1) Begin the transaction. This returns a transaction identifier that we can use with other statements.  
transactionId = system.db.beginTransaction(timeout = 5000)  
  
# 2) Now we can execute statements. Because we want these to run as part of the transaction, we need to  
include our identifier.  
query = "INSERT INTO orders (account_id, product_name) VALUES (?, ?)"  
args = [123, "Bananas"]  
system.db.runPrepUpdate(query, args, tx = transactionId)  
  
# 3) We can continue to add statements, but in this case we'll commit them. We could instead rollback if  
there was an issue with our previous statement.  
system.db.commitTransactions(transactionId)  
  
# 4) We're done, so close the Transaction.  
system.db.closeTransaction(transactionId)
```

Related Topics ...

- [Writing Basic SQL Queries](#)
- [system.db.runPrepUpdate](#)
- [Named Queries](#)
- [SQL Stored Procedures](#)

Common SQL Tasks

This section contains examples of items we identified as "common" tasks. When first starting out with [SQL in Ignition](#), many users are looking for examples in order to grasp concepts, or examples to possibly use in their projects. Additionally, this section aims to demystify some of the more complex or abstract tasks that our users may encounter when working with SQL in Ignition.

The examples in this section document several types of SQL tasks that may also touch many other areas of Ignition. While these examples are typically focused on a single concept or end result, they can easily be expanded or modified. In essence, they serve as a great starting point for users new to Ignition, as well as experienced users that need to get acquainted with a new or unfamiliar feature.

Below is a list of common SQL tasks related to using SQL in Ignition.

Filter Data in a Table

The [Filter Data in a Table](#) section explains how using a simple Dropdown box and making some modifications to the SQL query can allow users to dynamically filter data coming into the table to only show a specific subset of data, or all of the data. This section also includes variations of the SQL query, thus returning different filtered results to the table.

On this page ...

- [Filter Data in a Table](#)
- [Add Data to a Database](#)
- [Edit Data in a Database](#)
- [Refreshing SQL Data on a Component](#)
- [Editing Multi-Selected Rows from Table](#)
- [Storing PDF Files in a Database](#)
- [Simple Database Editor](#)

Add Data to a Database

A common method of [adding data to a database](#) from within a client is using a Button component that executes an SQL query. This section provides an SQL script showing how to collect relevant properties and insert them into a database, as well as how to setup your components on a window.

Edit Data in a Database

[Editing data in a database](#) is very similar to how we add data to a database. Users can edit data within a table on the screen in realtime, and the changes are then pushed back to the database table. This section goes over the different ways to edit data in the database depending if you're using a Table component or Power Table component.

Refreshing SQL Data on a Component

Sometimes, it may be unnecessary to have a table constantly updating and requerying the database for data. By using the `system.db.refresh` function, we can turn Polling "Off" on our query, and have a button that allows us to manually update the table with new data when the Button is pressed. Alternatively, we can add the `system.db.refresh` function at the end of a script to refresh the newly entered data automatically without pressing a button.

Editing Multi-Selected Rows from Table

Tables have the ability to select and [edit multiple rows](#) at a time. Oftentimes, this can be used to manipulate multiple rows simultaneously, such as deleting all of the selected rows at once.

Storing PDF Files in a Database

The database is a powerful tool and can allow you to store files such as PDFs in a database table. The [Storing PDF Files in a Database](#) section explores how we can take a PDF file and store it in the database table, as well as how to pull it back out and display it in the [Vision - PDF Viewer](#) component.

Simple Database Editor

It may be beneficial to have a frontend to a database table built right into an Ignition project that allows you to control the data in the table without having to go into the database's built-in frontend, such as MySQL workbench. In the [Simple Database Editor](#) section, you will learn how to build a simple database table editor on a window that can add, edit, and delete data from the database.

Related Topics ...

- [Writing SQL Queries](#)
- [Scripting Examples](#)
- [Common Reporting Tasks](#)

In This Section ...

Filter Rows in a Table

Creating the Components

Filtering table data by using a Dropdown box is possible if the **Table** component's **Data** property is bound to a SQL query. We can create a dynamic WHERE clause that will allow us to select all the data or a specific subset of it. To implement this solution, you must have a **Table** component and a **Dropdown List** component on a window. With a Dropdown component, we can create an option for each way of filtering the data, each with a unique value associated with it. An additional "all" entry can also be added, with its own unique value.

We can then use the value of the Dropdown to drive the query. When you select a different option from the Dropdown component the binding gets re-evaluated on the table's data binding resulting in the query executing the WHERE clause with the new parameters. You may want to turn Polling Mode off on the tables Data property binding in order to limit the periodic querying of the database.

Here we can use a value of 1 to show only entries which match that filter column value.

Pseudocode - Filtering for Area 1

```
SELECT * FROM table WHERE filterColumn = 1
```

While the following query will return everything, but will still evaluate the query's WHERE clause.

Pseudocode - Return All Rows

```
SELECT * FROM table WHERE 0=0
```

In this case 0=0 will always evaluate as true, therefore, the query will return every row.

We can combine the logic of each WHERE with an OR into a single query, and substitute in our Dropdown value. The SQL Query binding would then look something like this:

SQL - Where Clause Combined with OR Condition

```
SELECT * FROM table WHERE filterColumn = {dropdown value} OR 0 = {dropdown value}
```

This way, the WHERE clause will only be true when the filter column matches with our selected option, or it will return all rows, if we setup our "all" option to have a value of 0.

Example - Filtering Data on an Area Number

We can put together an easy example on how this might with data that stores what machines are in which areas. Our data table should look something like this:

id	machine_name	area_number
1	Conveyor	1
2	Press	2
3	Tank	1
4	Packer	3
5	Loader	3
6	Oven 1	3
7	Oven 2	2
8	Wrapper	1
9	Mixer	3

On this page ...

- [Creating the Components](#)
- [Example - Filtering Data on an Area Number](#)



INDUCTIVE
UNIVERSITY

Filter Data in Table

[Watch the Video](#)

10	Cold Storage	2
11	Dryer	2

We can then put together the query and components necessary to get this working:

1. Create a new Named Query. Set up security to fit your needs, and name it appropriately. For more information on creating Named Queries, see [Using Named Queries - Example](#).
 - a. Create a single Value type Parameter that is an Int2 data type. I called mine dropdownValue
 - b. Add in the combined query that we went over above, but use the machine table name and column names.

SQL - Selecting Values from a Table and Filtering on a Dropdown

```
SELECT * FROM machines WHERE area_number = :dropdownValue OR 0 = :dropdownValue
```

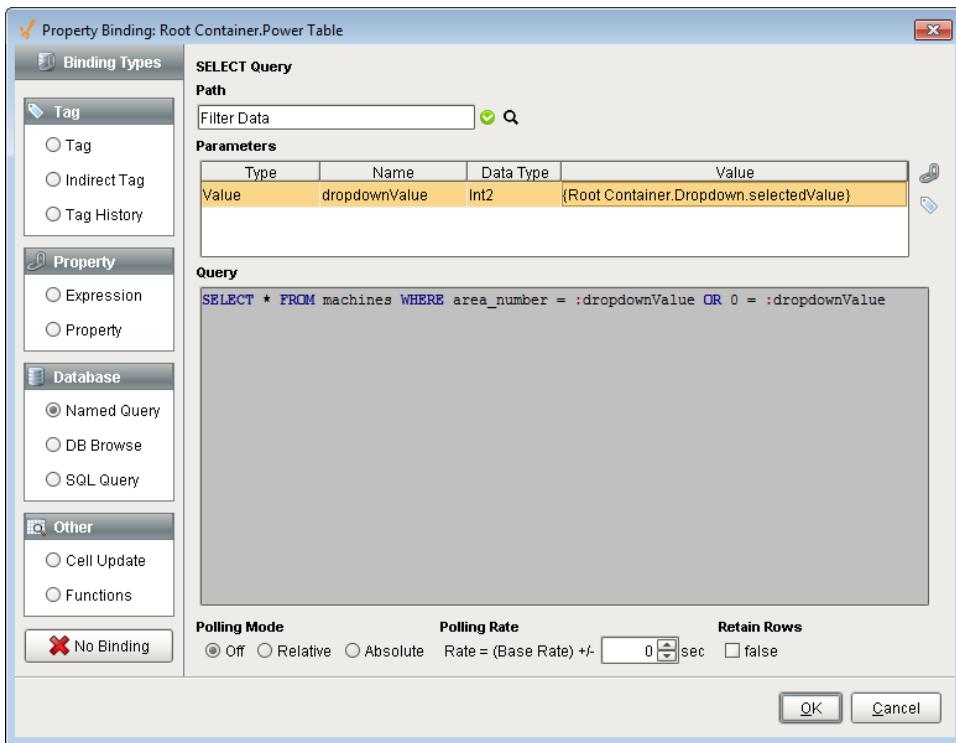
2. Create a new Main Window and add a Power Table component and a Dropdown component to the window.

- a. On the Dropdown's **Data** property, create a dataset that looks like this:

Value	Label
1	Area 1
2	Area 2
3	Area 3
0	All Areas

3. On the Power Table component, create a Named Query Binding on the **Data** property.

- a. Select the Named Query that was created in step 1.
- b. For the parameter, bind its value to the **Selected Value** property of the Dropdown component, and click the **OK** button.



4. Put the Designer into **Preview** mode and try out the Dropdown Filter. You will see that selecting a different value in the dropdown filters the data coming back so that only certain rows are shown.

id	machine_name	area_number
1	Conveyor	1
2	Press	2
3	Tank	1
4	Packer	3
5	Loader	3
6	Oven 1	3
7	Oven 2	2
8	Wrapper	1
9	Mixer	3
10	Cold Storage	2
11	Dryer	2

All Areas

Related Topics ...

- [Writing SQL Queries](#)

Inserting Data into a Database

Inserting Data on a Button Press

A common way to insert information into a database is to execute a SQL query after the user presses a Button. The button can run a script to collect the information it needs and then execute a SQL `INSERT` statement to push that data into the correct Database table. A script executed on the Button's `actionPerformed` event handler would collect the relevant properties and insert them into a database. The script would look like this:

Pseudocode - Collects Data and Insert into the Database

```
value1 = {component property reference}  
value2 = {tag value}  
value3 = {static value}  
  
query = "INSERT INTO table (col1, col2, col3) VALUES (?,?,?)"  
args = [value1, value2, value3]  
system.db.runPrepUpdate(query, args)
```

Notice that the script isn't limited from taking values from any one place, grabbing tag values and property values and inserting them into a table. It is important to note that using the `runPrepUpdate` function will require that certain [Client Permissions](#) have been enabled.

On this page ...

- [Inserting Data on a Button Press](#)
- [Example - Inserting Values](#)



INDUCTIVE
UNIVERSITY

Add Data to Database

[Watch the Video](#)

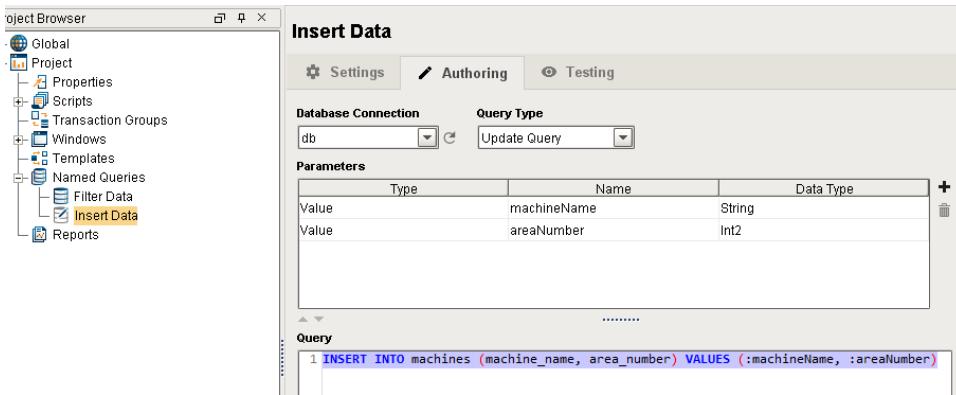
Example - Inserting Values

Say we have a table called `machines` in a database with three columns: an `id` column, a `machine_name` column, and an `area_number` column. We can build a query and a script that will insert the data into the database:

1. Create a new Named Query. Set up security to fit your needs and name it appropriately. For more information on creating Named Queries, see [Using Named Queries - Example](#).
 - a. Set the Query Type to Update Query.
 - b. Create two Value type Parameters.
 - i. The first will be a string data type and will be for the name of the machine so it can be called `machineName`.
 - ii. The second will be an Int2 data type and will be for the area number so it can be called `areaNumber`.
 - c. Create the insert query.

SQL - Inserting Values from Components on the Window

```
INSERT INTO machines (machine_name, area_number) VALUES (:machineName, :areaNumber)
```



2. On a Main Window, add a Button component, a Text Field component, and a Dropdown Component.

- a. On the Dropdown's Data property, create a dataset that looks like this:

Value	Label
1	Area 1
2	Area 2
3	Area 3

- b. On the Button's Text property, change the value to something like "Submit"



3. Right Click on the Button and select Scripting. Navigate to the Script Editor tab on the actionPerformed Event Handler.

- a. Copy in this script, which will pull in the value from the Dropdown and the Text Field and insert them into the table using the Named Query we built in step 1.

```
Python - Insert Values into a Database Table

# Grab the area number and machine name from the components we added to the window.
areaNum = event.source.parent.getComponent('Dropdown').selectedValue
machineName = event.source.parent.getComponent('Text Field').text

# A call to our Named Query, inserting the two parameters using dictionary syntax.
system.db.runNamedQuery("Insert Data", {"machineName":machineName, "areaNumber":areaNum})
```

4. Test it out by selecting an area, entering in a machine name, and clicking the submit button. You can check out the new rows using the [Database Query Browser](#) to select the values from that table.

Related Topics ...

- [Simple Database Editor](#)

Updating the Database through the Power Table

Updating Table Data

Table components can do more than show data from a database. A properly configured Table can make the data of the Table accessible to the Client and allow the user to edit the data in realtime. This page shows two examples using the Table and Power Table components editing data in a database in realtime.

Suppose your database had a Table like this:

id	UserName	FirstName	LastName	Notes
1	JS	John	Smith	Likes bikes
2	LJ	Luke	Johnson	Lives in town
3	PB	Peter	Burke	Enjoys cooking

On this page ...

- [Updating Table Data](#)
- [Using a Power Table Component](#)
- [Using a Table Component](#)



INDUCTIVE
UNIVERSITY

Edit Data in Database

[Watch the Video](#)

Using a Power Table Component

Let's create a script on a Power Table component and the **onCellEdited** extension function that will write to the cell in the database.



This function will not work using default [Client Permission](#) settings. The Legacy Database Access will need to be enabled for this to work.

1. Drag a Power Table component to your window, and bind the **Data** property of the Power Table using a SQL Query Binding Type and the query below.

SQL - Selects All Records

```
SELECT * FROM users
```

2. Right-click on the **Power Table** and select **Customizers > Table Customizer**.
3. Select the **Editable** checkbox for each of the columns that we want to edit (i.e., same as the above example). Note, the **id** column should not be editable. Click the **OK** button.
4. Right-click the **Power Table** and select **Scripting**.
5. Under the Extension Functions folder, select the **onCellEdited** function, and check the **Enable** box. We can put a script in here that will grab the **id** column value of the row that we modify, and then use that and the new value that we entered to update the Database table. The extension function makes this easy, because it provides variables for the column name of the row we are editing, the row number that was edited, as well as the new value. Paste the following script in the scripting area.

Python - Writes to a Cell in the Database

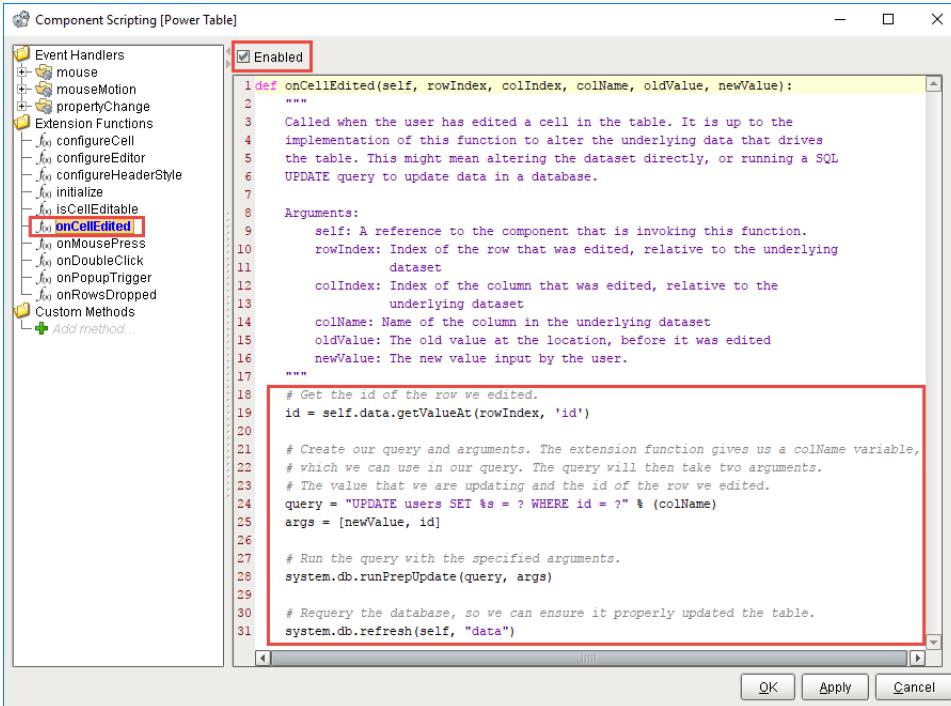
```
# Get the id of the row we edited.  
id = self.data.getValueAt(rowIndex, 'id')  
  
# Create our query and arguments. The extension function gives us a colName variable,  
# which we can use in our query. The query will then take two arguments.  
# The value that we are updating and the id of the row we edited.  
query = "UPDATE users SET %s = ? WHERE id = ?" % (colName)  
args = [newValue, id]
```

```

# Run the query with the specified arguments.
system.db.runPrepUpdate(query, args)

# Requery the database, so we can ensure it properly updated the table.
system.db.refresh(self, "data")

```



- Now, let's test it out. Put the Designer in **Preview Mode**, select the cell you want to edit, hit enter to commit the change, or tab to the next cell to make additional edits.

id	UserName	FirstName	LastName	Notes
1 JS		John	Smith	Likes bikes
2 LJ		Luke	Johnson	Lives in town
3 CP		Casey	Peters	Loves Ignition

Using a Table Component

Let's create another script like the example above, but this time we'll use a Table component's **cellEdited** event that will write to the cell in the database when we update it on the component. The script will be slightly different.



This function will not work using default [Client Permission](#) settings. The Legacy Database Access will need to be enabled for this to work.

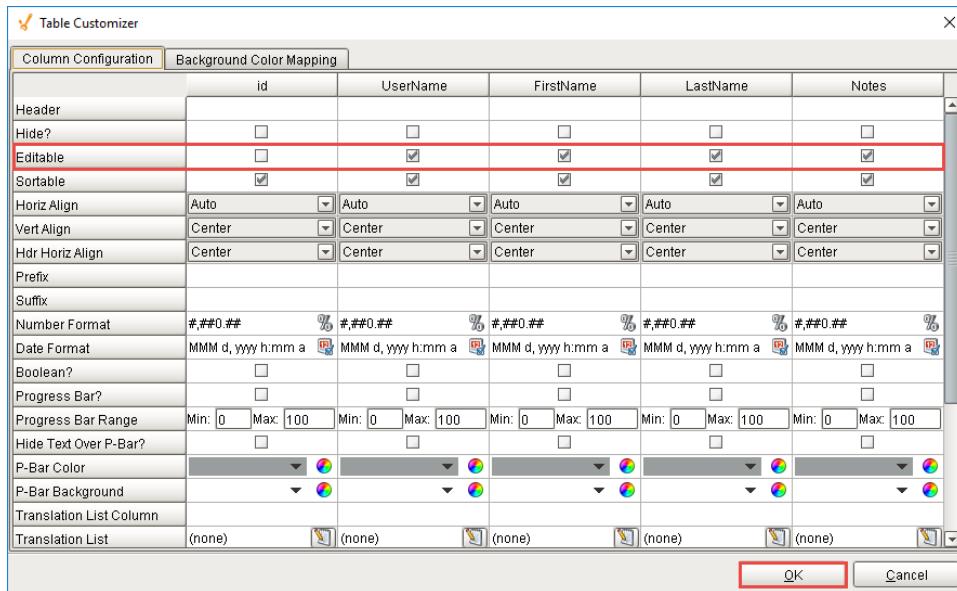
1. Drag a Table component to your window. With the Table component selected, bind the **Data** property of the Table using a SQL Query Binding Type and the query below.

```
SQL - Select All Records
SELECT * FROM users
```

id	UserName	FirstName	LastName	Notes
1	JS	John	Smith	Likes bikes
2	LJ	Luke	Johnson	Lives in town
3	PB	Peter	Burke	Enjoys cooking

2. Right-click on the **Table** and select **Customizers > Table Customizer**.

3. Select the **Editable** checkbox for each of the columns that we will want to edit. Note, the **id** column should not be editable. Click the **OK** button.



4. Right-click the **Table** and select **Scripting**.

5. Create a script in the **cell > cellEdited** event handler using the script provided below. In this script, we have variables that contain the row number that was edited, the column number that was edited, and the new value. Because we do not have the name of the column that was edited, we must first grab the list of columns using the **getColumnHeaders()** system function. We can then use the column number that was edited to find the name of the column that changed and use it in the query.

Python - Writes to a Cell in the Database When Component is Updated

```
# Get the id of the row we edited.
id = event.source.data.getValueAt(event.row, 'id')

# Get the header names of the dataset.
headers = system.dataset.getColumnHeaders(event.source.data)

# Build our Update query. The column name is substituted in from the column that was edited.
```

```

# The query will take two arguments. The value we are updating and the id of the row we are editing.
query = "UPDATE users SET %s = ? WHERE id = ?" % (headers[event.column])
args = [event.newValue, id]

# Run the query with the specified arguments.
system.db.runPrepUpdate(query, args)

# Requery the database, so we can ensure it properly updated the table.
system.db.refresh(event.source, "data")

```



6. Now, you're ready to test it out! Put the Designer in **Preview Mode**, select the cell you want to edit, hit enter to commit the change, or tab to the next cell to make additional edits.

id	UserName	FirstName	LastName	Notes
1	JS	John	Smith	Likes bikes
2	LJ	Luke	Johnson	Lives in town
3	EF	Ellen	Frank	Loves Ignition

Related Topics ...

- [Simple Database Editor](#)

Refreshing a SQL Query

Refreshing a Query in Vision

The SQL query that populates a property on a component will refresh its data periodically if the [Polling Mode](#) is set to either Relative or Absolute. However, there are times when you want the data to query the database once when the window is opened, and then retain explicit control over each subsequent refresh. The Table data can be requeried using `system.db.refresh`. There are two main ways of doing this: either placing the script on a button, or at the end of some code.

Refresh After Edits

If edits are being made to the Table data, or additional rows are being added, it can be a good idea to call the `system.db.refresh()` function after manipulating data. This way, the data in the Table will automatically refresh with the newly entered data, saving the user the hassle of clicking a refresh button.

Pseudocode - Refreshing a Table after an Edit

```
doWork()

system.db.runPrepUpdate("INSERT INTO table")

system.db.refresh(component, "propertyName")
```

On this page ...

- Refreshing a Query in Vision
 - Refresh After Edits
 - Refresh on a Button
- Refreshing a Query in Perspective



INDUCTIVE
UNIVERSITY

Refreshing SQL Data on Component

[Watch the Video](#)

Refresh on a Button

We can use a Button component with a script on it to refresh the component with the query binding on it. Typically the button is placed close by the query bound component, and will say something like "Refresh Data". The script is actually fairly simple, and can be placed on the `actionPerformed` event of the Button.

1. Drag a **Button** component onto a window that has a Power Table or Table component querying a Database table.
 - a. Change the Button's Text property to say "Refresh."
2. On the Power Table component, open the **data** property binding and set the polling mode to **off**.
3. Right click on the Button and select **Scripting**.
 - a. Select the **actionPerformed** Event Handler and navigate to the Script Editor tab.
 - b. Add in this script, which will force the Table's data property to refresh.

Python - Refreshing the Database when the Button is Pressed

```
# Will force the Power Table's Data property to run the query again.
system.db.refresh(event.source.parent.getComponent('Power Table'), "data")
```

4. Try it out by adding new data to the Database table and then clicking the Refresh Button.

Refreshing a Query in Perspective

The `system.db.refresh()` function does not work in Perspective. However, an additional component method has been added to handle the more general task of refreshing a binding. The method is called `refreshBinding()`, and can be called on any Perspective component:

```
self.refreshBinding("props.data")
```

The function takes a string as a parameter, the path to the property to be refreshed. More info on the `refreshBinding()` function can be found at [Perspective Component Methods](#).

Related Topics ...

- [system.db.refresh](#)
- [SQL Query Binding](#)
- [DB Browse Binding](#)
- [Named Query Bindings](#)

Editing Multi-Selected Rows from a Table

Editing Multiple Table Rows

A common user experience is to select multiple rows of a table and then edit all of those rows from the database at once, such as deleting all of them. When a user selects more than one row in a table, there is a special function called `getSelectedRows()` available on both the Table and Power Table components that returns the row indexes of the table as a Python list. This list can be iterated through in order to delete the selected rows from the database.

Example - Deleting Selected Rows

We can use the list of selected rows to delete them from the database. Start with a table in the Database table that looks like this:

id	machine_name	area_number
1	Conveyor	1
2	Press	2
3	Tank	1
4	Packer	3
5	Loader	3
6	Oven 1	3
7	Oven 2	2
8	Wrapper	1
9	Mixer	3
10	Cold Storage	2
11	Dryer	2

On this page ...

- [Editing Multiple Table Rows](#)
- [Example - Deleting Selected Rows](#)



INDUCTIVE
UNIVERSITY

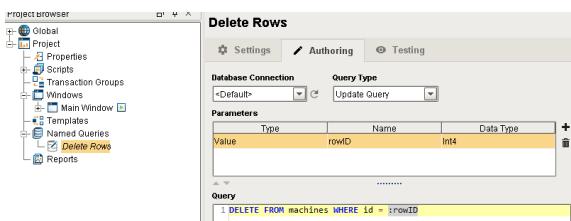
Deleting Multi-Selected Rows from Table

[Watch the Video](#)

1. Create a new Named Query that will be used to delete rows of data. [Setup Security](#) to fit your needs, and name it appropriately. For more information on creating Named Queries, see [Using Named Queries - Example](#).
 - a. Set the Query Type to **Update Query**.
 - b. Create a single Value type Parameter that is an Int4 data type. This will hold the id of the row, so we can name it `rowID`.
 - c. Create the Delete Query.

SQL - Delete Row with Matching ID

```
DELETE FROM machines WHERE id = :rowID
```



2. Create a second Named Query as in step 1, but this one will be used to select the data into a table.
 - a. Set the Query Type to Query, no parameters, and add a basic select query.

SQL - Selecting from a Table

```
SELECT * FROM machine
```

3. On a new Main Window, add a Power Table component and a Button component.
4. On the Power Table's **Data** property, set up a Named Query binding to the Select Query that was made in step 2.
 - a. Ensure that the Selection Mode property is set to **Multiple Interval**.
5. On the Button, change the Text property to say "Delete".
 - a. Right click on the Button and select Scripting.
 - b. Select the actionPerformed Event Handler and navigate to the Script Editor Tab.
 - c. Here we need to call the **getSelectedRows()** function on the Power Table to determine what rows are selected, and then loop through those to grab the value of the id column in each row and delete the row based on that id.

Python - Looping Through the Selected Rows and Deleting Them

```
# Get the data from the table and assign it to the variable
# called data.
data = event.source.parent.getComponent('Power Table').data

# Get the rows of the data that the user has currently
# highlighted.
rows = event.source.parent.getComponent('Power Table').
getSelectedRows()

# Iterate through each row of the list that is associated
# with the rows variable.
for row in rows:

    # Get the value associated with the current row and
    # the column called "id".
    id = data.getValueAt(row, "id")

    # Run the query to delete from the database, where
    # tableName is the name of the database table.
    system.db.runNamedQuery("Delete Machine Rows",
    {"rowID":id})

    # Refresh the table data.
    system.db.refresh(event.source.parent.getComponent('Power
    Table'), "data")
```

6. You can test it out by putting the Designer into Preview Mode, selecting a few rows, and then clicking the Delete Button.

Related Topics ...

- [Updating the Database through the Power Table](#)
- [Refreshing a SQL Query](#)

Storing Files in a Database

Storing and Displaying Files in a Database

Ignition can store different types of files into a database by storing the raw file bytes into a special database column. Ignition can also pull these file bytes out and display certain files within a Client.



Each database has different column types that are used to store files, so it is important to check with your database documentation to see which data type the column would need to be set to for it to accept file bytes. For example, in MySQL, the data type that accepts PDF bytes is a **LongBlob** data type, so you will need to set the PDF Data column to the **LongBlob** data type. MS SQL accepts the **Varbinary** data type, so you'll need to set the PDF Data column to a **Varbinary** data type.

On this page ...

- [Storing and Displaying Files in a Database](#)
- [Example - PDF File](#)
 - [Uploading PDF Files to the Database](#)
 - [Displaying PDF Files from the Database](#)

Example - PDF File

One of the most common file types that is stored is PDF files. This allows you to store each PDF file within the central database where each client will have access to it, instead of placing the file in a shared drive that all Client computers have access to.



INDUCTIVE
UNIVERSITY

Storing Files in a Database

[Watch the Video](#)

Uploading PDF Files to the Database

With a simple script on a Button component, we can store a PDF file into the database so that any user can view it later. This part does not use a Named Query, because the Named Query Parameters do not have a data type that allows us to pass in the raw file bytes. We can instead use [system.db.runPrepUpdate](#) to call a query from the script.

This example requires that you have a table with a byte array column in it. For example: MySQL uses the BLOB data type and MSSQL uses the varbinary() data type.



This function will not work using default [Client Permission](#) settings. The Legacy Database Access will need to be enabled for this to work.

1. Add a Button component to a new Main Window.
2. Change the Text property to say Add File.
3. Right click on the Button and select Scripting.
 - a. Navigate to the Script Editor Tab of the actionPerformed Event Handler. Here we can put a script that will grab the file bytes using the file path and the [system.file.readFileAsBytes](#) function, and then insert that into the database, along with a user selected file name.

Python - Uploads PDF Files to a Database Using a Button

```
# Find the path to the PDF file.
path = system.file.openFile("pdf")

# Check to ensure that the user actually selected a filepath.
if path != None:

    # Read the file as bytes.
    data = system.file.readFileAsBytes(path)

    # Ask the user to enter in a filename.
    # Will grab just the filename and extension from the path as a suggestion.
```

```

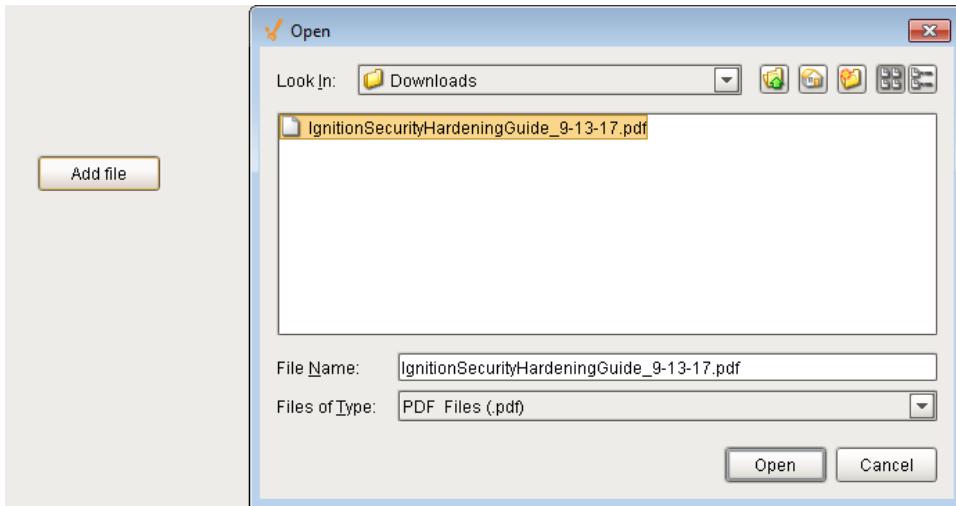
name = system.gui.inputBox("Enter a name for the file", path.split('\\')[-1])

# Check to ensure that the user entered a name for the file.
if name != None:

    # Insert the data and name into the database.
    system.db.runPrepUpdate("INSERT INTO files (fileName, fileBytes) VALUES
(?,?)", [name,data])

```

4. Test out the script by putting the Designer into Preview Mode and clicking the Button.
 a. First, select the file to load.



- b. Then enter in a file name.



Displaying PDF Files from the Database

Ignition can render a PDF document inside the **Vision - PDF Viewer** component, which is a part of the **Reporting Module**. To view PDF files in the Client, your Ignition server must have the Reporting Module installed. Once the module is installed, you can load the bytes from the database into the PDF Viewer component.

1. Create a new Named Query that will be used to select the file names and ids for a Dropdown selection. Set up security to fit your needs, and name it appropriately. For more information on creating Named Queries, see [Using Named Queries - Example](#).
 - a. With no Parameters, add a query to select all the files in our files table.

SQL - Selecting all Files
<pre>SELECT id, fileName FROM files</pre>

2. Create a second Named Query, same as in step 1. This will be used to grab the file name and bytes after the user has chosen a file.
 - a. Add a single Value type Parameter, "fileID" that will be an Int4 data type.
 - b. Add the query to select the file name and bytes based on the selected ID.

SQL - Selecting a File based on an ID

```
SELECT fileName, fileBytes FROM files WHERE id = :fileID
```

3. On the Main Window, add a Dropdown List component and a PDF Viewer component.
4. On the Dropdown List component, add a Named Query binding to the Data property.
 - a. Set the binding to the Named Query that was made in step 1.
 - b. Place a small refresh Button next to the Dropdown that will refresh this query. See [Refreshing a SQL Query](#) for more information for how to refresh on a button.
5. Right click on the Dropdown List component and select Scripting.
 - a. Select propertyChange and navigate to the Script Editor tab.
 - b. This script will take any new selected value and use it in the Named Query we made in step 2 to get the file name and bytes. We can then load the bytes into the PDF Viewer.

Python - Displays PDF Files from a Database Using the PDF Viewer

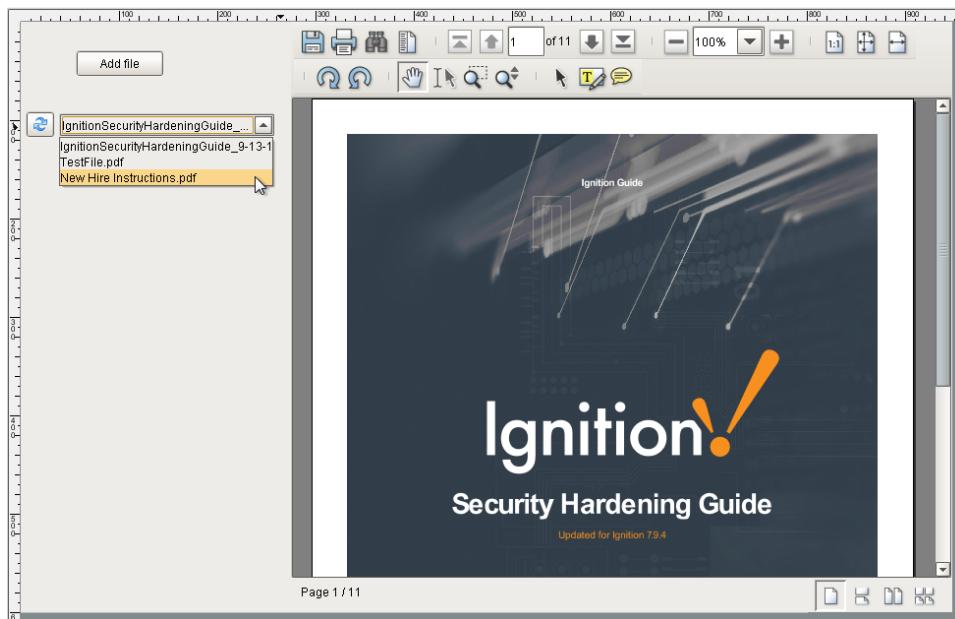
```
# Check to see if the property that changed was the Selected Value property.
if event.propertyName == "selectedRow":

    # Run the query to grab the file name and bytes using the new selected ID value.
    data = system.db.runNamedQuery("Read File", {"fileID":event.newValue})

    # Grab the file bytes and name from the same row.
    bytes = data.getValueAt(0, "fileBytes")
    name = data.getValueAt(0, "fileName")

    # Load the bytes into the PDF Viewer component.
    event.source.parent.getComponent('PDF Viewer').loadPDFBytes(bytes, name)
```

6. Place the Designer into Preview Mode, and try selecting one of the stored files.



Related Topics ...

- [Vision - PDF Viewer](#)

Simple Database Editor

Building a Database Table Editor

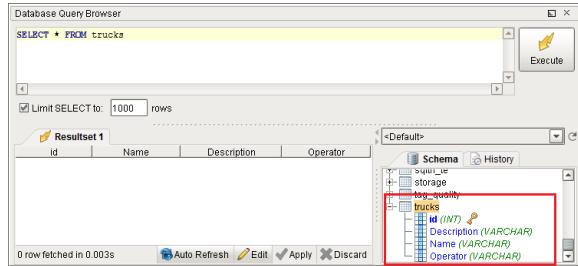
Building a Simple Database Table Editor in Ignition is actually quite easy. Using a few simple components and some scripting, we can easily make a window that will allow users to add, edit, and delete the data from a particular database table. Before we get started, there is a table that has already been created in a database that is used for this example. You can either create this table and follow along, or alter the example to fit your table.

The table is called **trucks**, and it has four columns.

id	Name	Description	Operator
----	------	-------------	----------

- The **id** column is our primary key, it is non null, and it auto increments
- All other columns are varchar(45)

That is it. There is no need to add any data, as we can add data when testing our tool!



Adding the Components

This example consists of a Table component that will display the data, Button components to add and delete rows from the database table, Text Fields where we can enter in new values, and an Update Details Button that will push those values to the database table.

Let's start off by going to the **Designer** and adding the necessary components to the window.

1. Add a **Power Table** component to the window.
 - a. We want to query the table in the database that we made earlier, so let's add a **SQL Query Binding** on the **Data** property of the Table.
 - b. Add the following code under the **SELECT Query** area.

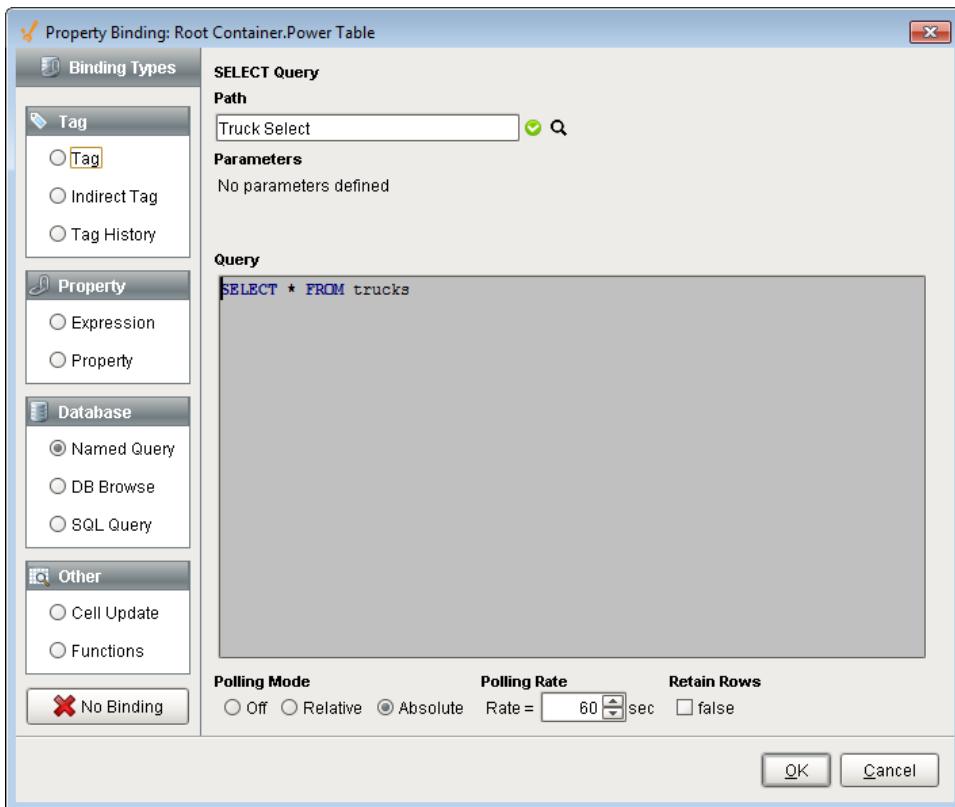
SQL - Selects all Trucks

```
SELECT * FROM trucks
```

- c. Set the **Polling Mode** to **Absolute** and the **Polling Rate** to **60 seconds**, so that it periodically updates if left open.
- d. Since we want to use the secure Named Query system, click the convert to Named Query button under the polling mode, and call the query "Truck Select"

On this page ...

- [Building a Database Table Editor](#)
 - [Adding the Components](#)
 - [Add a Truck](#)
 - [Delete a Truck](#)
 - [Populate Text Fields](#)
 - [Update Data](#)

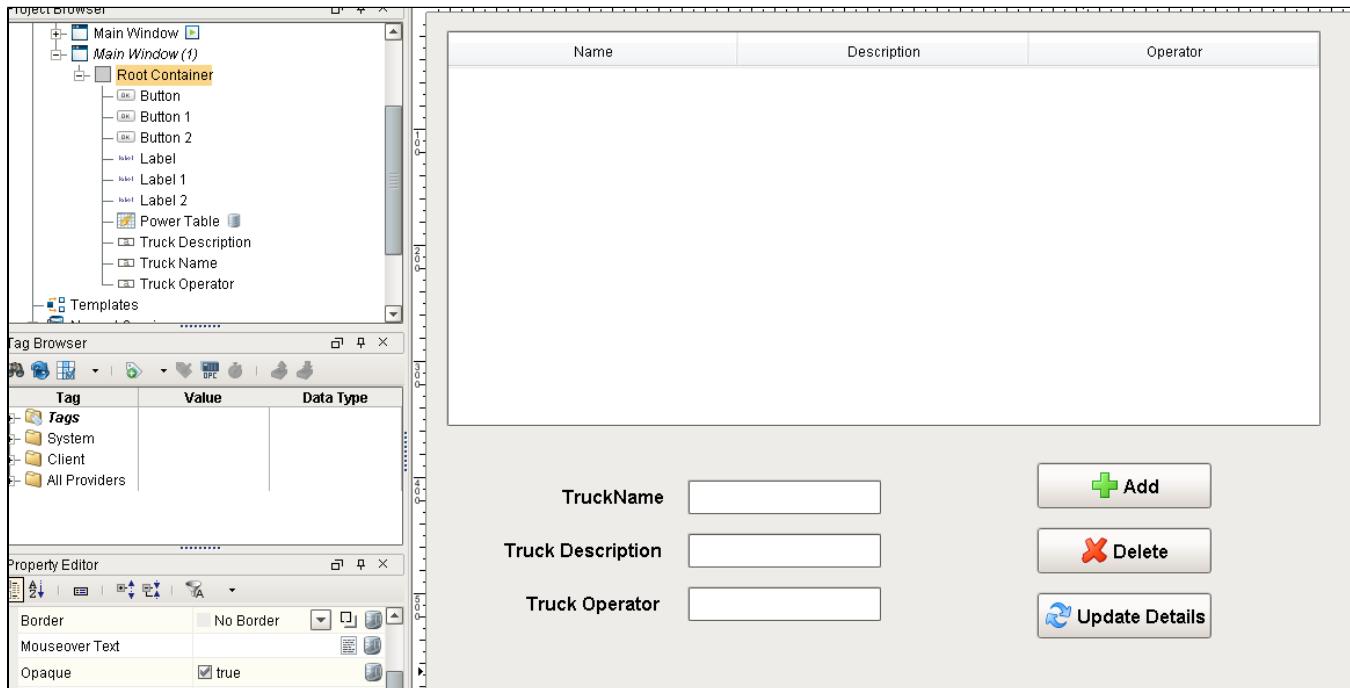


- e. With the Table still selected, open the **Table Customizer** by right clicking on the Power Table and going to **Customizers > Table Customizer**.
- f. Set the **Hide** property of the **id** column to **true**. We don't need to show that column to our users. Click **OK** to close the Table Customizer.
- g. In the **Property Editor**, make sure the **Selection Mode** property of the Power Table is set to **Multiple Interval**.

- 2. Add three **Button** components to the right of the **Table**.
 - a. Change the **Text** property on one Button to say "**Add**".
 - b. Change the **Text** property of the second Button to say "**Delete**".
 - c. Change the **Text** property of the final Button to say "**Update Details**".
 - d. Optionally, you can also add an image to the **Image Path** of each button such as a green plus and a red X, and resize them as necessary.

- 3. Add three **Label** components and three **Text Field** components under the **Table**.
 - a. Pair up a Label with each Text Field.
 - b. Rename the Text Fields:
Truck Name, Truck Description, Truck Operator.
 - c. Change the **Text** property of each Label to match the name of their corresponding Text Fields.

Now we have our window configured with all of the components we need. Next we can start adding scripts that will alter the components.

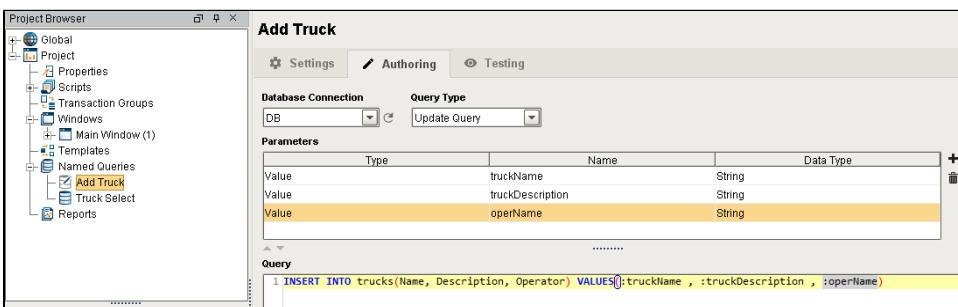


Add a Truck

First, we can add a script to our **Add** button that will allow us to add a row to the Table. We can use the Text Fields that we added so the user can enter in values to insert into the table.

1. Create a new Named Query which we will use to insert new data. Set up [security](#) to fit your needs and name it appropriately. For more information on creating Named Queries, see [Using Named Queries - Example](#).
 - a. Set the Query Type to Update Query.
 - b. Create three Value type Parameters.
 - i. The first will be a string data type and will be used for the name of the truck so it can be called "truckName".
 - ii. The second will be a string data type and will be used for the description of the truck so it can be called "truckDescription".
 - iii. The third will also be a string data type and will be used for the operators name so it can be called "operName".
 - c. Create the insert query.

SQL - Adding a Truck			
<pre>INSERT INTO trucks(Name, Description, Operator) VALUES(:truckName , :truckDescription , :operName)</pre>			



2. Back on our window with components, right click on the **Add Button** and select Scripting. Select the **actionPerformed** Event Handler and navigate to the **Script Editor** tab.
 - a. Here we can add some code that will pull the values from the three Text Field components and then use those values in the Named Query we made in step 1.

Python - Adds a New Blank Row and Refreshes the Table	
---	--

```

# Grab the values from the Text Fields.
name = event.source.parent.getComponent('Truck Name').text
description = event.source.parent.getComponent('Truck Description').text
operator = event.source.parent.getComponent('Truck Operator').text

# Use those values in the Add Truck Named Query.
system.db.runNamedQuery("Add Truck", {"truckName":name, "truckDescription":description,
"operator":operator})

# Refresh the table to immediately bring in the new row.
system.db.refresh(event.source.parent.getComponent("Power Table"), "data")

```

3. Now we can test out our script and query by putting the Designer into **Preview Mode**, adding some data to the **Text Fields**, and clicking the **Add Button**. You should see a row populate the table. Try adding a few rows to fill our table with data. Make sure you have the designer Communication Mode set to Read/Write to test. Go back to **Design Mode** before moving on.

Name	Description	Operator
Green Monster	It is so green!	Sarah
Little Truck	The smallest truck.	Joe
Big Truck	This truck is big!	James
Fire Truck	Puts out fires. Does not transport fire.	Dale
Ice Cream Truck	Sells ice cream and various other treats.	Amy
Toy Truck	Transports toys, not a truck that is a toy.	Jane
Blue Truck	A truck that is painted blue.	Anne

TruckName Add

Truck Description Delete

Truck Operator Update Details

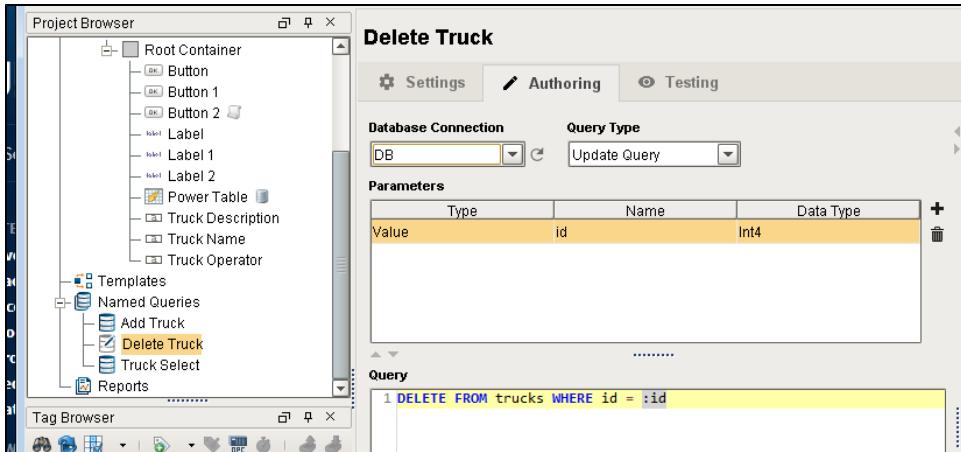
Delete a Truck

Now that we can add rows, let's add a script to our **Delete button** that will delete rows. Since our users can select multiple rows, our script needs to take that into account, and delete all of the rows that the user selected. It will also have a **Message Box** popup if there are no rows selected, informing the user that they need to select at least one row. We also want to add a confirmation before the rows actually get deleted.

- Create a new Named Query which we will use to delete a row of data. Set up **security** to fit your needs and name it appropriately. For more information on creating Named Queries, see [Using Named Queries - Example](#).
 - Set the Query Type to Update Query.
 - Create one Value type Parameters.
 - It will be a Int4 data type and will be used for the id of row so it can be called "id".
 - Create the delete query.

SQL - Deleting a Truck

```
DELETE FROM trucks WHERE id = :id
```



2. Back on our window with components, right click on the **Delete Button** and select Scripting. Navigate to the **Script Editor** tab on the **action Performed** Event Handler.

a. Here we can add some code that will delete the user's selected rows. First we grab the Power Table's selected rows, and check to make sure rows are actually selected. If there are selected rows, then we have a popup that confirms the user wants to delete rows, and it shows the number of rows that will be deleted. Once the user confirms, the script will then loop through the selected rows and delete each one.

```
Python - Deletes the Row or Rows

# Get the selected row or rows.
selRows = event.source.parent.getComponent('Power Table').getSelectedRows()

# Check to see that a row is actually selected.
if len(selRows) > 0:

    # If a row is selected, ask for confirmation before deleting the rows.
    if system.gui.confirm("Are you sure you want to delete " + str(len(selRows)) + " row(s)?", "Are You Sure?", 0):

        # If the user gave confirmation, we need to loop through all of them.
        for row in selRows:

            # For each selected row, we need to get the value of the id column in that row.
            id = event.source.parent.getComponent('Power Table').data.getValueAt(row, "id")

            # Use the id of the row to delete it from the database table.
            system.db.runNamedQuery("Delete Truck", {"id":id})

            # Refresh the table after deleting all selected rows
            # to immediately remove the selected rows from the Power Table.
            system.db.refresh(event.source.parent.getComponent('Power Table'), "data")

        # If the user said no to the delete.
    else:
        system.gui.messageBox("User canceled the delete.", "Delete Canceled")

    # If no row is selected, have a popup message that asks the user to select a row.
else:
    system.gui.messageBox("Please select at least one row.", "Select A Row")
```

3. Now we can test our **Delete button** by putting the Designer into **Preview Mode** and clicking the **Delete button**. When one or more rows are selected and the **Delete button** is pressed, a popup will confirm we want to delete the rows before deleting them. (You can select multiple rows with **Shift-Click** or **Control-Click**). When no rows are selected and the **Delete button** is pressed, a **Message Box** will pop up, informing the user to select a row first. You can deselect rows in the Table by clicking on the Table and pressing the **Escape** key. Be sure to go back to **Design Mode** before moving on.

Name	Description	Operator
Green Monster	It is so green!	Sarah
Little Truck	The smallest truck.	Joe
Big Truck	This truck is big!	James
Fire Truck	Puts out fires. Does not transport fire.	Dale
Ice Cream Truck	Sells ice cream and various other treats.	Amy
Toy Truck	Transports toys, not a truck that is a toy.	Jane
Blue Truck	A truck that is painted blue.	Anne

Are You Sure?
 Are you sure you want to delete 2 row(s)?

TruckName	<input type="text" value="Blue Truck"/>	 Add
Truck Description	<input type="text" value="A truck that is painted blu"/>	 Delete
Truck Operator	<input type="text" value="Anne"/>	 Update Details

Name	Description	Operator
Green Monster	It is so green!	Sarah
Little Truck	The smallest truck.	Joe
Fire Truck	Puts out fires. Does not transport fire.	Dale
Toy Truck	Transports toys, not a truck that is a toy.	Jane
Blue Truck	A truck that is painted blue.	Anne

TruckName  **Add**
Truck Description  **Delete**
Truck Operator  **Update Details**

Populate Text Fields

Now that we can add and delete rows from our Table, we need a way to edit the data contained in them. However, before we can create a script on our **Update Details button**, we first need to populate the Text Fields with the current values of the selected row. We can do this by adding a simple expression to each of the text fields.

1. On the Truck Name Text Field, add an expression binding to the Text property.
 - a. The expression will grab the dataset and will grab the value at the selected row of the Name column. The expression is wrapped in a try so that if no rows are selected, it will return an empty string.

Expression - Grabbing the Selected Row's Name Value

```
try({Root Container.Power Table.data}[{Root Container.Power Table.selectedRow}, "Name"], "")
```

2. On the Truck Description Text Field, add an expression binding to the Text property.

- a. The expression will grab the dataset and will grab the value at the selected row of the Description column. The expression is wrapped in a try so that if no rows are selected, it will return an empty string.

Expression - Grabbing the Selected Row's Description Value

```
try({Root Container.Power Table.data}[{Root Container.Power Table.selectedRow},
"Description"] , "")
```

3. On the Truck Operator Text Field, add an expression binding to the Text property.

- a. The expression will grab the dataset and will grab the value at the selected row of the Operator column. The expression is wrapped in a try so that if no rows are selected, it will return an empty string.

Expression - Grabbing the Selected Row's Operator Value

```
try({Root Container.Power Table.data}[{Root Container.Power Table.selectedRow}, "Operator"],
"")
```

4. Click on a row to see the fields below populate. Press the ESC key to empty the selection.

Name	Description	Operator
Green Monster	It is so green!	Sarah
Little Truck	The smallest truck.	Joe
Fire Truck	Puts out fires. Does not transport fire.	Dale
Toy Truck	Transports toys, not a truck that is a toy.	Jane
Blue Truck	A truck that is painted blue.	Anne

TruckName

Truck Description

Truck Operator

+ **Add**

✖ **Delete**

🕒 **Update Details**

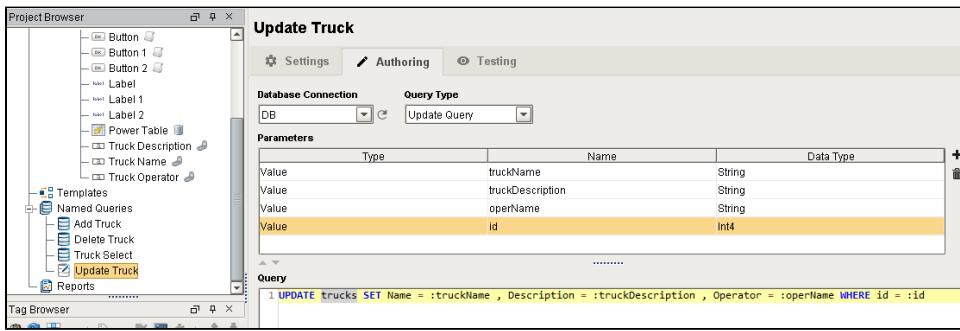
Update Data

The last part of setting up this window is to add a script to the **Update Details** button that will pull the data from the Text Fields and insert it into the selected row of the database Table. Additionally, the script will check to ensure that a row selected, and that only one row is selected. We wouldn't want to confuse users who may have accidentally selected multiple rows at a time, since our button will only update a single row.

- Create a new Named Query which we will use to update a row of data. Set up **security** to fit your needs and name it appropriately. For more information on creating Named Queries, see [Using Named Queries - Example](#).
 - Set the Query Type to Update Query.
 - Create four Value type Parameters.
 - The first will be a string data type and will be used for the truck name of row so it can be called "truckName".
 - The second will be a string data type and will be used for the truck description of row so it can be called "truckDescription".
 - The third will be a string data type and will be used for the operator name of row so it can be called "operName".
 - The fourth will be a Int4 data type and will be used for the id of row so it can be called "id".
 - Create the update query.

SQL - Deleting a Truck

```
UPDATE trucks SET Name = :truckName , Description = :truckDescription , Operator = :operName
WHERE id = :id
```



2. Back on our window with components, right click on the **Update Button** and select Scripting. Navigate to the **Script Editor** tab on the **action Performed** Event Handler.

a. Here we can add some code that will update the users selected row with data pulled from the Text Fields. First we check to ensure that only one row is selected. Next the script asks them to confirm that the user wants to update the selected row. If the user confirms, the script then pulls in the values of all three Text Fields and uses those values along with the selected rows id to update the database table with new data. It will then refresh the table after updating so that the new data can be brought into the table.

```
Python - Deletes the Row or Rows

# Check to make sure only one row is selected.
if len(event.source.parent.getComponent('Power Table').getSelectedRows()) == 1:

    # If a row is selected, ask for confirmation before updating the row.
    if system.gui.confirm("Are you sure you want to update the selected row?", "Are You Sure?", 0):

        # Grab the values from the text fields.
        name = event.source.parent.getComponent('Truck Name').text
        description = event.source.parent.getComponent('Truck Description').text
        operator = event.source.parent.getComponent('Truck Operator').text

        # Grab the selected row.
        selRow = event.source.parent.getComponent('Power Table').selectedRow

        # Using the selected row, we need to get the value of the id column in that row.
        id = event.source.parent.getComponent('Power Table').data.getValueAt(selRow, "id")

        # Run a query that will update the values of the row matching the id.
        system.db.runNamedQuery("Update Truck", {"truckName":name, "truckDescription":description, "openName":operator, "id":id})

        # Refresh the table to immediately show the updated data.
        system.db.refresh(event.source.parent.getComponent('Power Table'), "data")

        # If the user said no to the update.
        else:
            system.gui.messageBox("User canceled the update.", "Update Canceled")

    # If there more than one row selected, or no rows selected.
    else:

        # Have a popup asking the user to select one row.
        system.gui.messageBox("Please select one row to edit.", "Select One Row")
```

3. Now that we added this last bit of code, we can test it by selecting a row, adding data to each **Text Field**, and clicking the **Update Details** button. You should see the data populate the row of the Table, since the table in the database has been updated. Additionally, you should notice how the Text Fields will automatically update when selecting a new row. Try adding a few rows of valid data to your Table.

Name	Description	Operator
Green Monster	It is so green!	Sarah
Little Truck	The smallest truck.	Joe
Fire Truck	Puts out fires. Does not transport fire.	Dale
Toy Truck	Transports toys, not a truck that is a toy.	Jane
Yellow Truck	A truck that is painted yellow.	Abby

TruckName

Truck Description

Truck Operator

+
Add

✖
Delete

✎
Update Details

Related Topics ...

- [Inserting Data into a Database](#)
- [Updating the Database through the Power Table](#)
- [Editing Multi-Selected Rows from Table](#)
- [Named Query Caching](#)

Basic SQL Troubleshooting

Learning when something is wrong with your database queries and learning how to resolve the problem is key to getting the best possible use out of the database connection to Ignition. Typically, the biggest problem that users face with queries is the data taking too long to load on the window, but an error in a query can also be difficult to track down. This section details what to look for when the query fails to execute, as well as what to do when facing slow queries and how to optimize them within the project to help return data to the window as quickly as possible.

Error Message Box

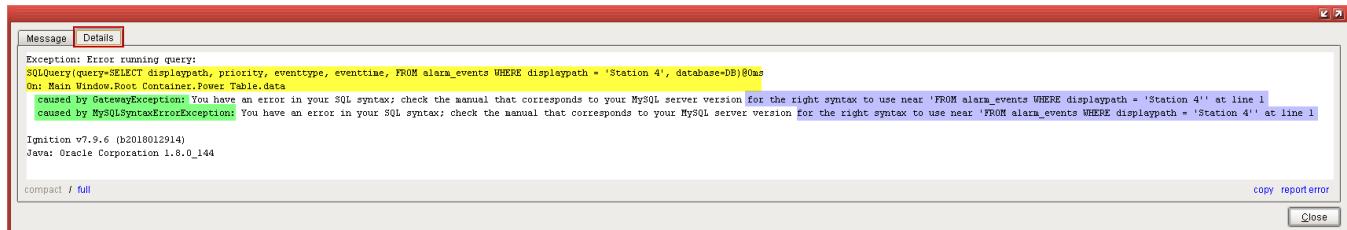
If a query fails to execute, chances are there was an error box that popped up. Resist the urge to close it! The Error Box usually contains a lot of useful info that can help troubleshoot why the query is failing. When looking at the error message, there are a few things to look for. First, you will want to click on the **Details** tab in the upper left corner of the Error Box. This will bring up the details of the error, which contains the information we need to track down the error.

In the image below, you can see the yellow highlighted part shows us what our query is, as well as where it is executing from. This helps to ensure we are looking at the correct error message for the query we are trying to fix. Next, you want to look for the words '**caused by**' in the text of the error details, which tell us the reason for the error. You can see in the image below, there are two of '**caused by**' messages highlighted in green. The first one is from the Gateway, and the second one is from the database (in this case, a MySQL database).

After locating the '**caused by**', the following message will help to pinpoint what the error is. In this example, we have an error with our syntax in our SQL statement. The most helpful portion of the error is at the end of the message highlighted in purple. This lets us know where the syntax error is in the SQL statement. The trick is to look immediately before the quoted query. In the image below, it starts to quote the original query with '**FROM alarm_events WHERE.....**', so in my original query, I need to look at what was right before the '**FROM**'. You can see in my original query highlighted in yellow, there is a comma right before the '**FROM**', which is incorrect, as the last column in the **SELECT** statement should not have a comma after it.

On this page ...

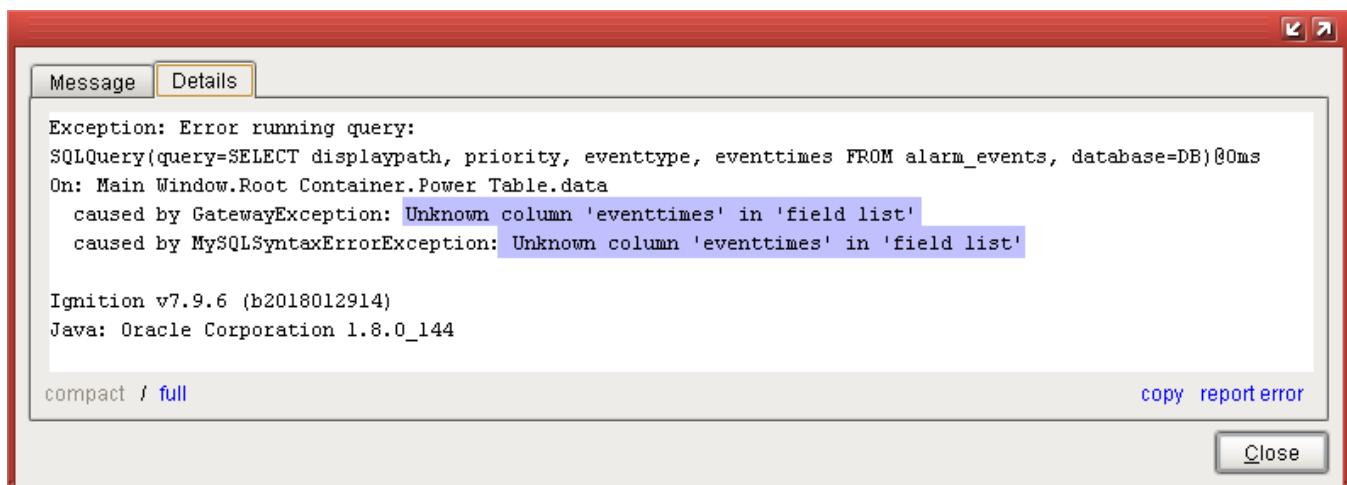
- Error Message Box
 - Other Common SQL Errors
- Checking the Database Connection
- Testing Query Results
- Checking for Slow Queries



Other Common SQL Errors

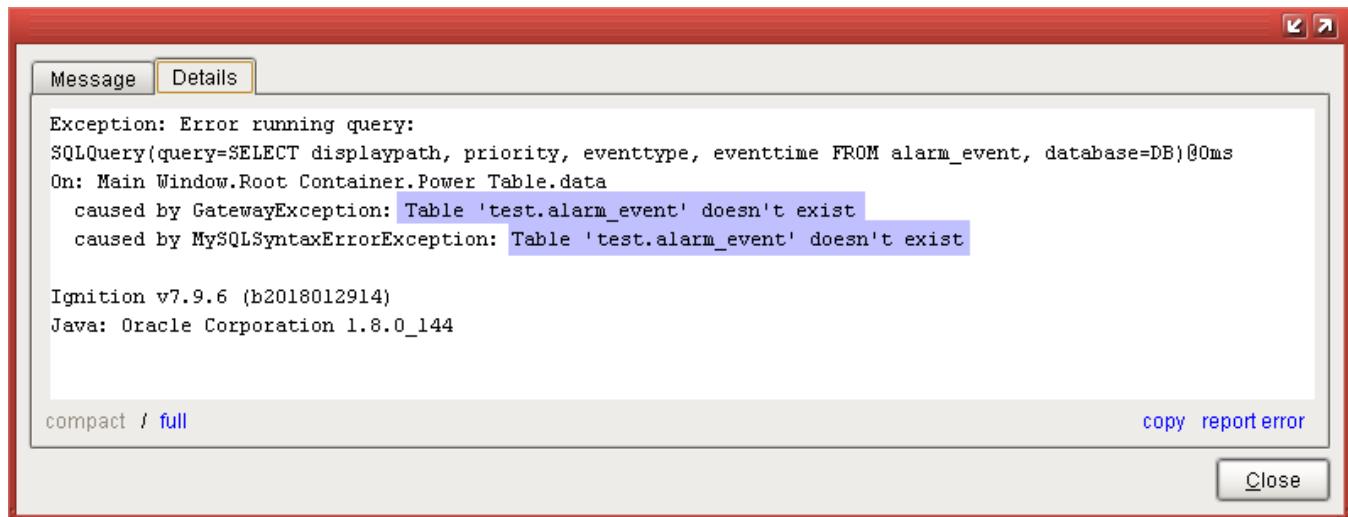
Unknown Column

Typically, the name of the column is wrong because of a misspelling. In this case, the correct spelling was 'eventtime', but there was an accidental 's' added to the name.



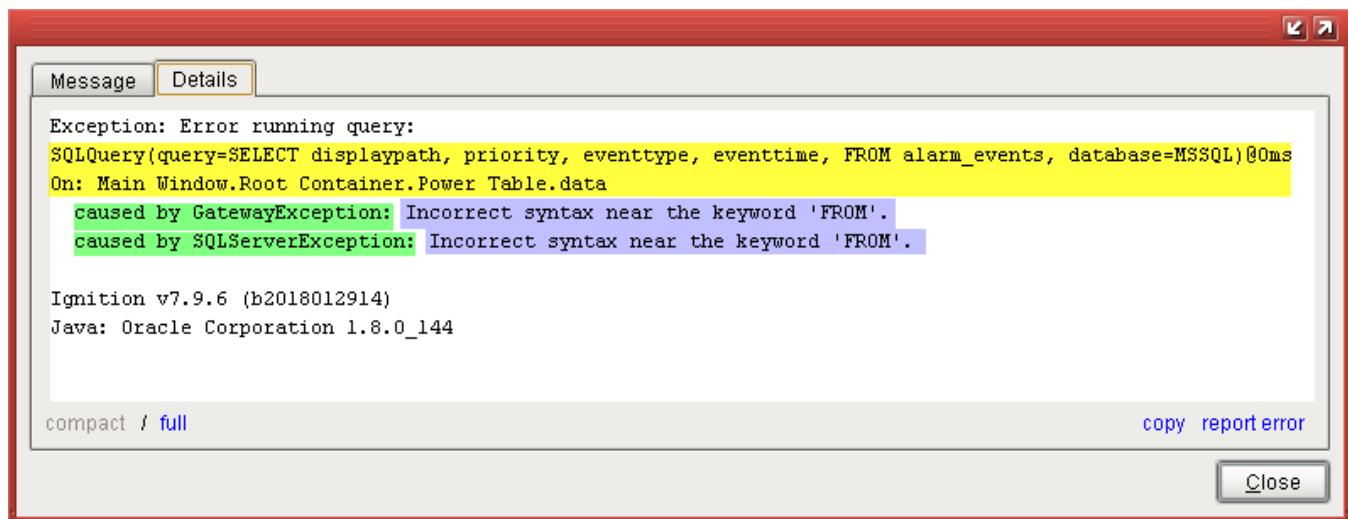
Unknown Table

If the table is not found in the database, it may also be because of a misspelling. Here, the correct table name should be 'alarm_events'.



Any Database Error

The previous error examples have all been from a MySQL database, but the same principles apply to any database. Simply locate the 'caused by', and look at the message afterwards. In this MSSQL syntax error, instead of displaying everything after the syntax error, it only displays the part immediately after. The problem in this query is an extra comma before the word 'FROM'.



Checking the Database Connection

Checking the Database Connection in the Gateway can also be useful to ensure there is a valid connection and there is nothing blocking the execution of queries. The [Status](#) page of the Gateway can be used to determine if queries are taking too long, while the Database Connections page can be used to alter the settings of the database to better handle the number of queries running from the Gateway to the database. See the [Slow Queries](#) page for more information on changing these settings.

Testing Query Results

Using the [Database Query Browser](#) is a good way to test out a query before actually running it on a component. This can help you see what results will be returned so that the query can be modified to narrow down how the query should be formatted.

Checking for Slow Queries

There are several ways that slow running queries can cause improper behavior. See [Slow Queries](#) for more details.

[Related Topics ...](#)

- [SQL in Ignition](#)

[In This Section ...](#)

Slow Queries

Slow running queries can be a big problem. Not only can data take a long time to display on the screen, but it can end up slowing down the whole client. Here, we take a look at some of the things that you can do when your project has a slow running query.

Identify the Slow Query

The first step in dealing with slow queries is identifying which query is actually running slowly. It is typically obvious when a window is opened and a component takes time before database data is displayed. Chances are, the query is on that component somewhere. However, it may also be a good idea to check the queries that are being run against that [database connection in the Gateway](#). From the Gateway webpage, navigate to **Status Connections Databases** to see a list of all database connections. Clicking on the **Details** button to the right of the database connection will show all of the currently running queries, the most recent long running queries, as well as some basic metrics for that connection. Here, you can get a good idea of any queries that may be a little slower than the others.

On this page ...

- [Identify the Slow Query](#)
- [Execute Against the Database Management Software](#)
- [Fast Query in the Database Management Software](#)
 - [Check the Database Connection](#)
 - [Check Currently Executing Queries](#)
- [Slow Query in the Database Management Software](#)

The screenshot shows the Ignition software interface in the 'Status' mode. The left sidebar includes sections for Systems (Overview, Performance, Alarm Pipelines, Gateway Scripts, Modules, Redundancy, Reports, SFCs, Tags, Transaction Groups), Connections (Databases, Designers, Devices, Gateway Network, Store & Forward, OPC Connections, OPC-UA Server, Vision Clients), and Diagnostics (Execution, Logs, Threads). The 'Databases' section is currently selected. The main content area shows 'Database Stats' with a circular gauge indicating 1 connection and 0 queries per second. Below this is a histogram of query execution times. The 'Active Queries' section shows a single query: 'SELECT sleep(60)' started 'a few seconds ago'. The 'Longest Recent Queries' section lists several queries with their execution times:

Query	Duration	Started
UPDATE sqlth_sce SET `end_time`=? WHERE `scid`=? and `rate`=? and `start_time`<=? and `end_time`>=?	7 ms	15 minutes ago
SELECT sleep(60)	59.841 s	6 minutes ago
SELECT `tagid`,`intvalue`,`floatvalue`,`stringvalue`,`datevalue`,`t_stamp`,`dataintegrity`		

Execute Against the Database Management Software

The next step in identifying the type of slow query we are dealing with is to run the query directly within the database management software. By cutting out Ignition, we can determine if the query is actually running slowly, or if there is a problem within Ignition that is making the query run slow.

Fast Query in the Database Management Software

If the query runs quickly in the database management software, then there are a few things we can take a look at that may help out.

Check the Database Connection

First, check to make sure the database connection is valid, and there are no warnings associated with it. You may want to go into the connection settings for that database and take note of the value of the **Max Active** property, which determines the maximum number of active connections to the database as well as the **Max Wait** property, which is the number of milliseconds to wait for a connection to come available. To get to the database connections settings, go to **Configure** page of the Gateway webpage and select **Databases Connections**, locate your database, and hit the **edit** button. Here, you can check the database settings. Open **Advanced Properties** at the bottom of the screen, and you'll find the **Max Active** property as well as the **Max Wait** property.

Check Currently Executing Queries

Back in the **Status** section of the Gateway Webpage on the **Database Connection** page mentioned above, we can see a list of currently running queries, as well as how many of the active connections are being used. If the max number of connections is being used, it may be that there are so many queries running that each query needs to wait for an active connection to open up. If this is the case, you may want to increase the amount of active connections to the database or take steps to [reduce the query load on the database](#).

Slow Query in the Database Management Software

If the query also runs slow within the database management software, then the query is just a slow query. Unfortunately, nothing within Ignition can speed up the execution of that query, so you would want to instead take a look at what the query is doing. If the query is pulling in lots of data, you can try breaking the query down into smaller queries, or writing the query in a more efficient way.

For particularly large tables, it may also be helpful to add an index to one of the table's columns. Indexes are something that the user can't see, but help the database speed up data retrieval. However, adding an index to a table will increase the amount of time that an update to the table takes, because the index also needs to be updated. For this reason, it is recommended to only make indexes on columns that are frequently searched against.

To make an index, most database management software have built in interfaces that allow you to customize the index on each table. An index can be made for a single column, or a combination of columns in the table. Talk to your Database Administrator about adding or updating table indexes.

Alternately, you may need to take a look at the database system as a whole. As the size of the database grows, you may need to update the hardware resources available to it. If the database is installed on a server with another system like Ignition, be aware that although the two systems are now sitting next to each other, they now have to share the hardware resources available to them, which may cause issues for both systems. In many cases, it is often better to have the database run on a separate server, which gives it ample room to grow.

Related Topics ...

- [SQL Query Volume Optimization](#)
- [Connections - Databases](#)

SQL Query Volume Optimization

Overview

Leveraging a SQL Database can drastically improve the quality of a project, but improper database consideration while designing can lead to poor performance later on. This page contains some best practices and considerations when incorporating SQL queries.

Optimizing Individual Queries

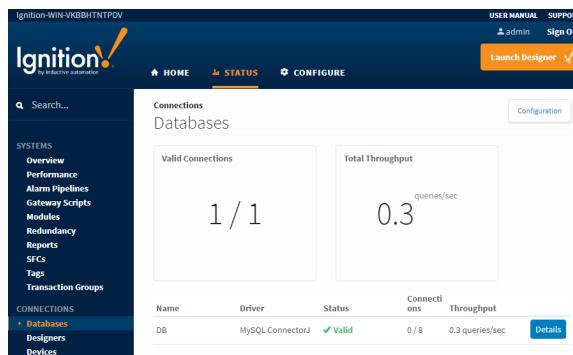
Optimizing individual queries to run faster and be more efficient is very difficult to do properly and can vary widely depending on a number of factors such as the way the tables are setup in the database and what data is being pulled out. Because each database can vary widely from another, there isn't any general way of improving the efficiency of your queries. Instead we recommend becoming more familiar with the SQL language as well as how the data is setup in your system and exactly what data you want to retrieve. This knowledge can help you build better queries.

Your company may also have a database administrator who would manage the database system for your company and would be familiar with figuring out the best way of retrieving data. They may be able to help you retrieve the data that you need.

Always Think Large

Consider how many instances of a query may be running at any given point in time. A single [SQL Query Binding](#) will be called for each instance of the window that is open, so if 50 clients are all looking at the same window, 50 separate queries will be called. If the binding is configured to poll, then 50 queries will poll at the rate specified for this single binding. This is already a fair amount of work without factoring in other systems, such as Tag Historian.

To provide context, you can always check the [Status Section](#) of the Gateway Webpage to check current throughput of each database connection.



Use Cached Named Queries When Possible

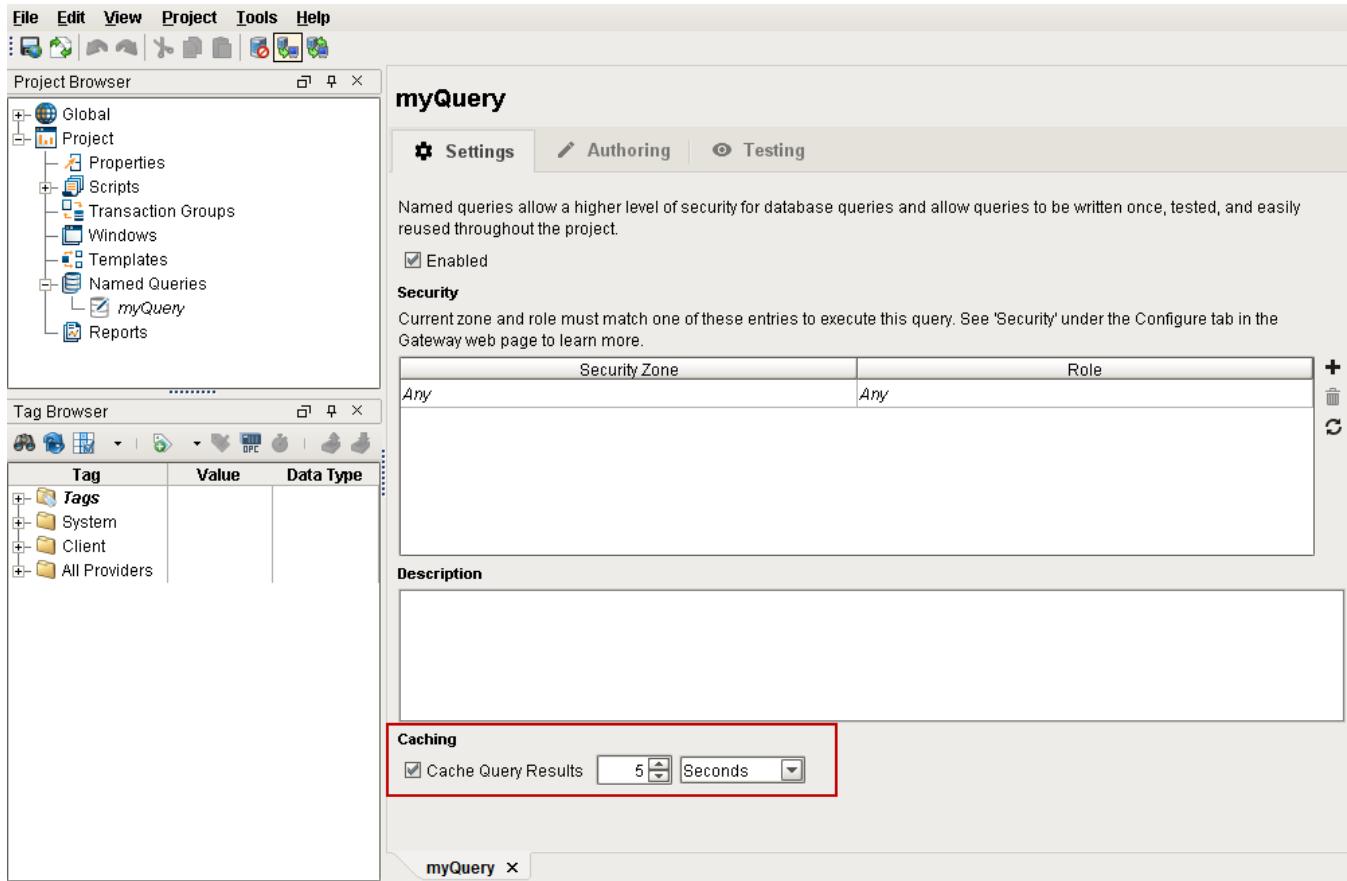
Most resources in Ignition that can request a query will not cache the results for use by other resources: a SQL Query Binding that returns a result set will only do so for the one component, and can't be utilized by other resources in the same project. Thus, if two clients have the same window, the same query must be fired twice for both bindings to receive information from the database which is wasteful.

[Named Queries](#) are the exception. They can [cache the resulting dataset](#) for use by other resources in the project, as well as other instances of the same project. In the previously mentioned scenario, one client would trigger the Named Query to execute, and the other client would simply utilize the cached result set, reducing the number of queries running against the database.

If your project contains queries that poll slowly, or results sets that aren't frequently modified, then a Named Query with caching enabled is an efficient alternative to a SQL Query or DB Browse binding.

On this page ...

- Overview
- Optimizing Individual Queries
- Always Think Large
- Use Cached Named Queries When Possible
- Use the Expression Language to Consolidate Multiple Queries
- Restrict the Number of Query Tags in a UDT
 - Single Database Tables
 - Multiple Database Tables



Use the Expression Language to Consolidate Multiple Queries

If multiple resources (such as multiple Tags, or multiple components) need separate values from the same database table, or a window contains multiple components that are all querying data from the same table, such as multiple Numeric Labels, it may be more efficient to have a single query run and fetch the large portions of the table, and then distribute the individual values to each component. This typically involves having some property or Tag query for several rows of data from a database table, and then using expression bindings or Expression Tags to retrieve individual values from the query.

Individual values may be retrieved from a dataset via the Expression Language: either an Expression binding on a property, or an Expression Tag. Here are two commonly used approaches to extracting a single value from a dataset in the Expression Language:

- The [Expression Language's Dataset Access](#) syntax. You may want to use the `try()` function in case the dataset is empty:

Pseudocode - Expression Language Dataset Access Syntax

```
{dataset}[rowIndex,columnIndex]
```

- The `lookup` function:

Pseudocode - Expression Language Lookup Function

```
lookup({dataset}, lookupValue, noMatchValue)
```

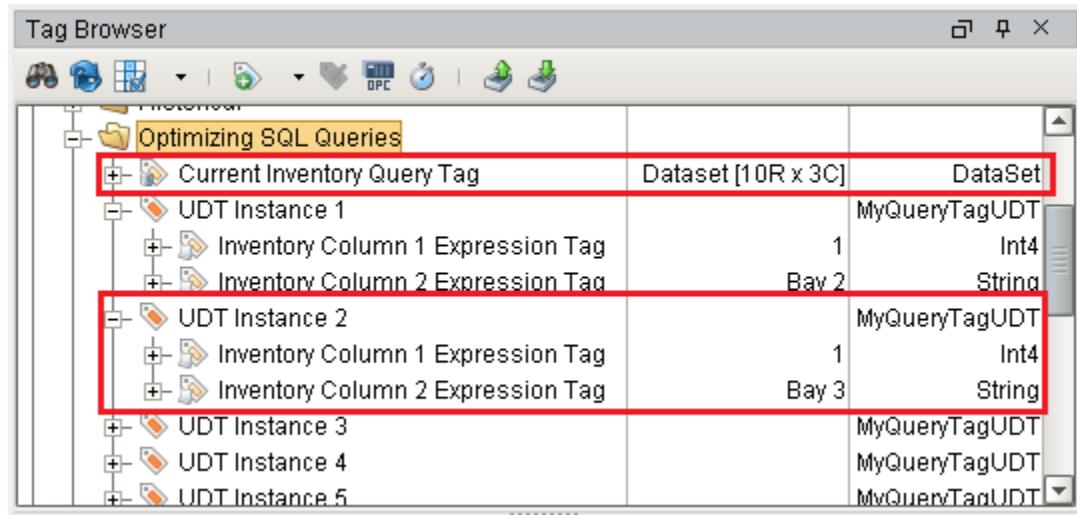
Restrict the Number of Query Tags in a UDT

Each Query Tag in a UDT will run a separate query per instance of the UDT. Assuming a scan class of one second, if a UDT definition contains 5 Query Tags, and there are 5 instances of that UDT, then there will be 25 queries executing every second.

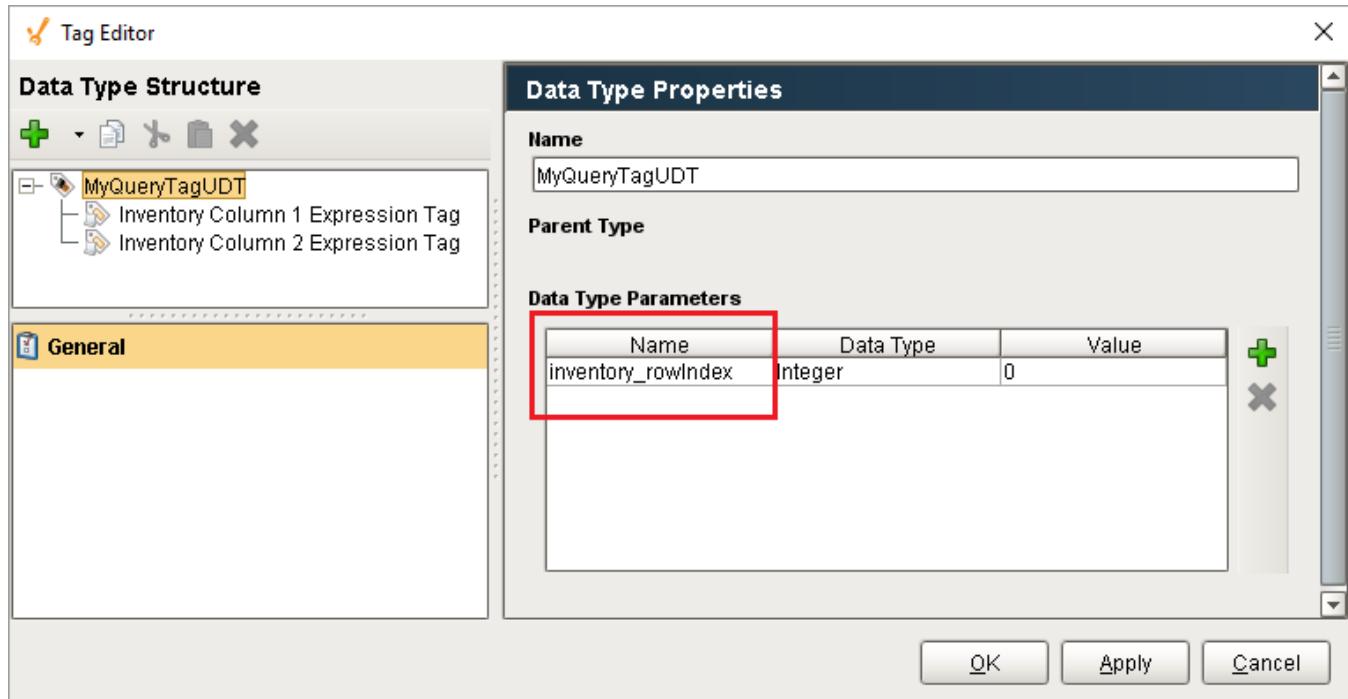
As mentioned on this page, the Expression Language can be used to reduce load on the database if multiple Query Tags are retrieving data from the same database table. Furthermore, UDT parameters can be utilized in the Expression Tags, so new UDT Instances can easily be configured to look up the appropriate values.

Single Database Tables

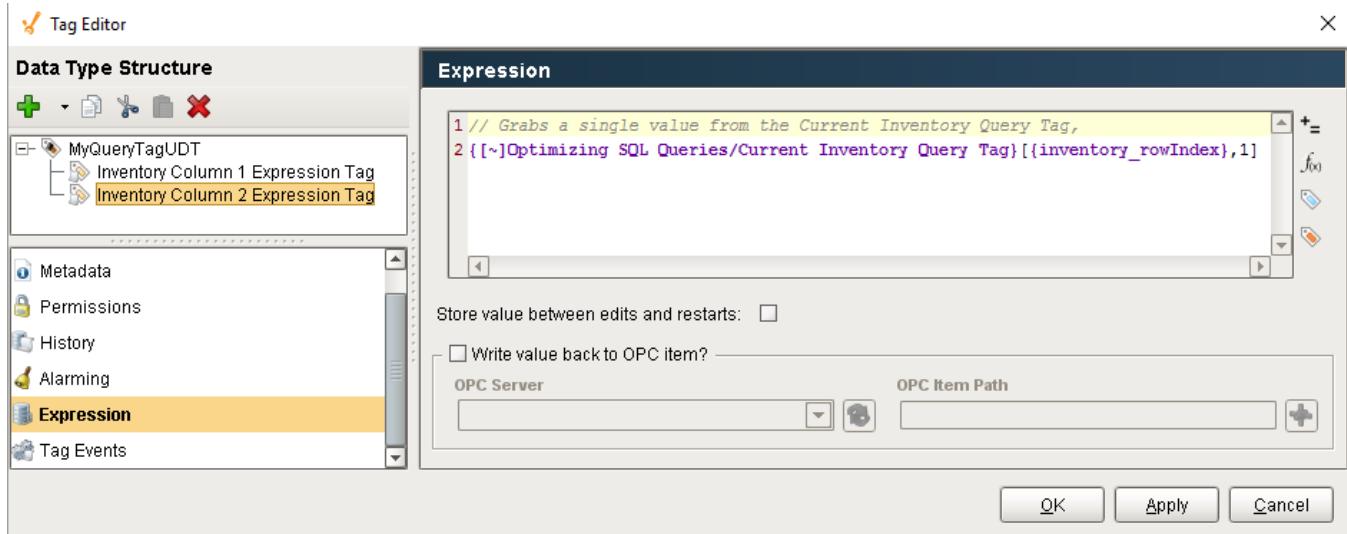
Below, the Tag named **Current Inventory Query Tag**, as the name implies, is a Query Tag retrieving multiple rows of data from a database table. We see that the highlighted **UDT Instance 2** contains two members: **Inventory Column 1 Expression Tag** and **Inventory Column 2 Expression Tag**, which are simply Expression Tags that are referencing individual cells from the **Current Inventory Query Tag**.



The UDT definition can use a parameter to specify an individual row in the Query Tag that each instance should focus on.



Each Expression Tag could use an expression like the following to look up individual values:



Multiple Database Tables

To add values from a separate database table, we simply need a separate Query Tag. In the image below, a new Tag named **Maintenance Query Tag** has been added, which is querying from a separate Database table. To incorporate this new data into our UDT instances, new Expression Tags have been added (**Maintenance Column 1 Expression Tag**, and **Maintenance Column 2 Expression Tag**) that simply reference specific values in the new Query Tag. Now, regardless of how many UDT instances exist in the Tag provider, we only have two Tags that are executing queries against the database.

For each separate table, we need to incorporate a single new Query Tag to collect all of the rows we want to show, add index parameters to the UDT definition, and add Expression Tags to our UDTs.

Tag Browser			
+ Historical			
- Optimizing SQL Queries			
+ Current Inventory Query Tag	Dataset [10R x 3C]	DataSet	
+ Maintenance Query Tag	Dataset [9R x 6C]	DataSet	
+ UDT Instance 1		MyQueryTagUDT	
+ UDT Instance 2		MyQueryTagUDT	
+ Inventory Column 1 Expression Tag	1	Int4	
+ Inventory Column 2 Expression Tag	Bay 3	String	
+ Maintenance Column 1 Expression Tag	1	Int4	
+ Maintenance Column 2 Expression Tag	item1	String	
+ UDT Instance 3		MyQueryTagUDT	

Related Topics ...

- [Named Query Caching](#)
- [Slow Queries](#)

Scripting

What Is Scripting?

Most of the time when we talk about "scripting" in Ignition we are talking about Python scripting, or writing code in the Python language. Python is a general purpose programming language that was developed in the early 1990s and has gained significant popularity in the 2000s. It is extremely readable, elegant, powerful, and easy to learn. As an added bonus, it gracefully interacts with Java, giving programmers an extremely powerful tool when paired with Ignition, which is written in Java.

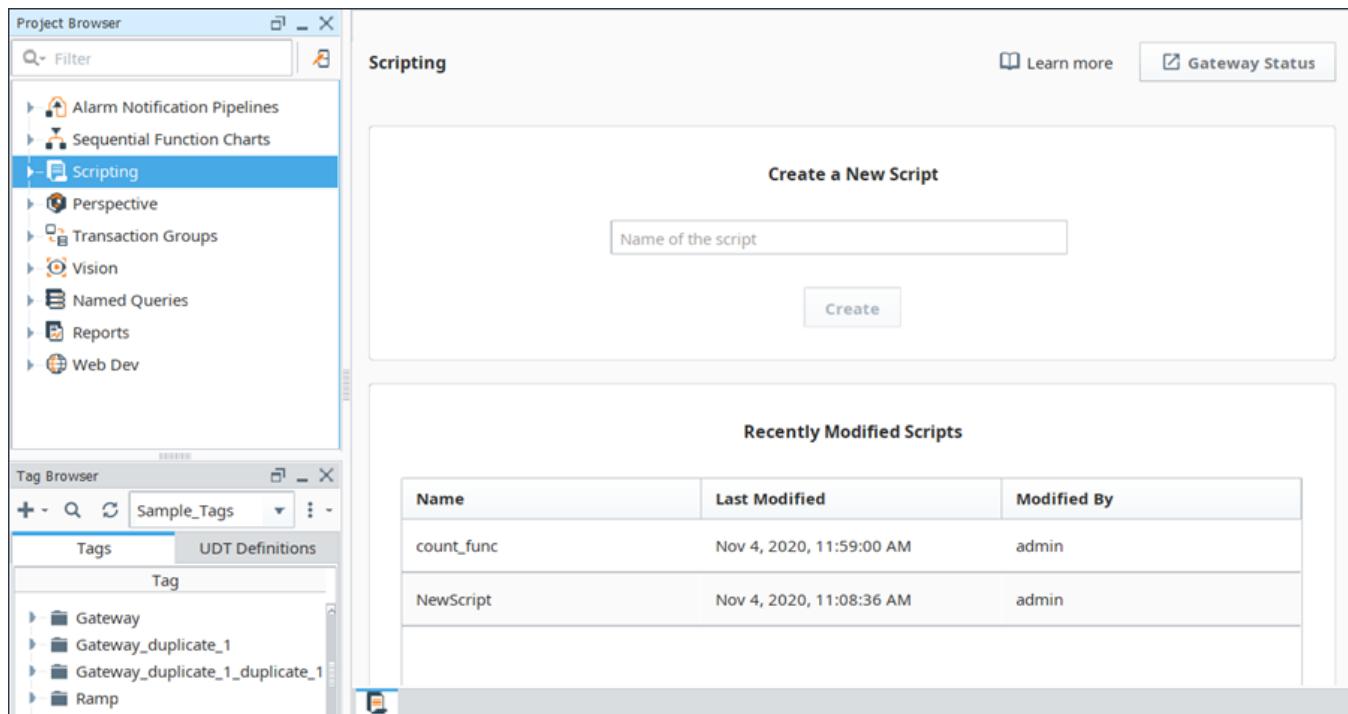
The Scripting Welcome tab will appear when you have either the Scripting window or one of its children open. It allows you to create a script, and once you click Create, it immediately opens the window so you can start writing your script. It's a quick and efficient way to get right to your scripting tasks. At a glance, the Scripting Welcome tab will also show you any recently modified scripts along with the date it was modified and who modified it. You can even double click on a recently modified script and open it.

The Scripting Welcome tab provides a quick way to create a new script and update existing ones. The **Welcome Tab** should appear when the user has either Scripting, or one of its children items selected.

In Ignition, you will be mixing the core Python language with references to other components and a variety of our built-in [system functions](#).

On this page ...

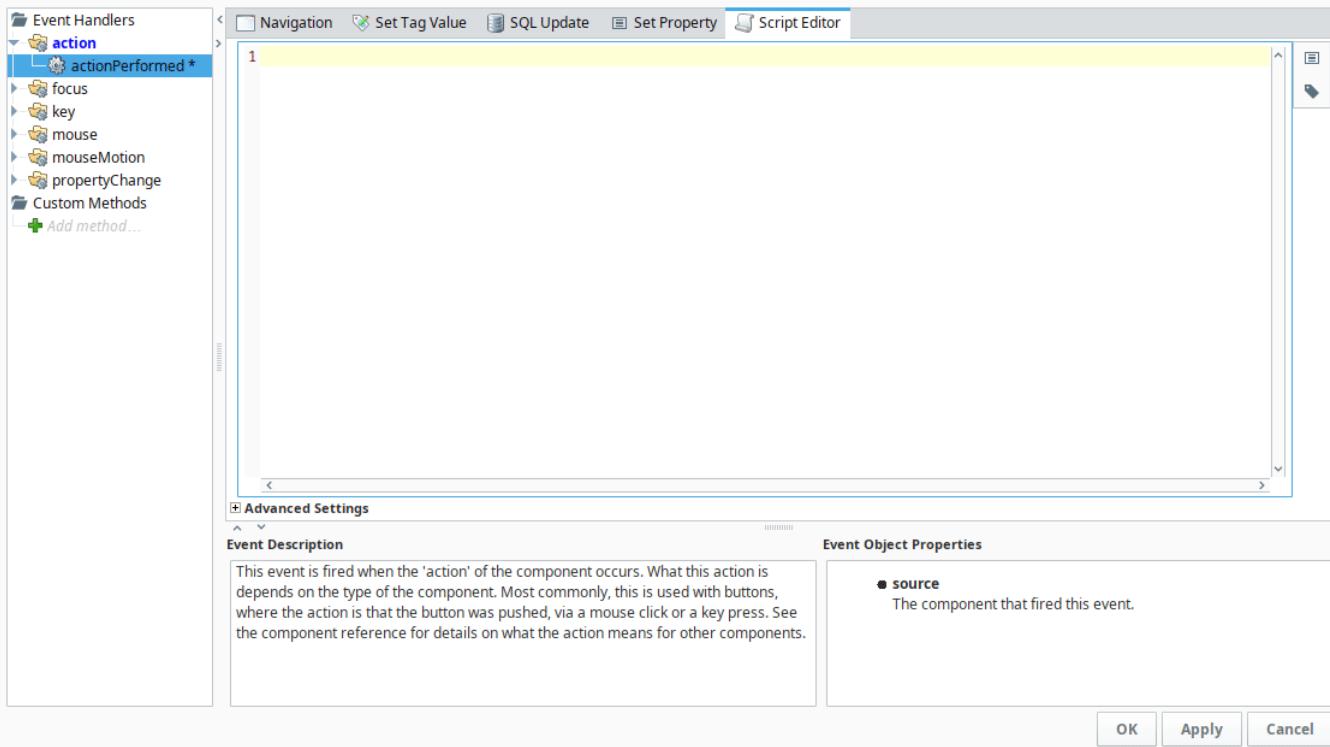
- [What Is Scripting?](#)
- [Is it Easy to Use?](#)
- [Where Is Scripting Used?](#)



Is it Easy to Use?

Luckily, Python is a simple language to get started with. Using it in an event-driven system takes away a lot of the extra code that normally makes programming time consuming. For those that are already familiar with scripting (and those of you that are learning), we have a huge list of functions inside Ignition to do some of the common tasks in a single line of code. These [System Functions](#) are available while typing. Just start with "**system**"

and press "ctrl-space" to see a list of available functions. The list will filter itself as you continue typing.



Where Is Scripting Used?

Python is used in many places in Ignition. Each location has its own events that trigger your scripts to run, and add functionality to your projects in different ways.

- **Components** - Add actions to components like buttons, customize the look and feel of charts and tables, and even setup a custom navigation schema.
- **Tags** - Create a script that runs on a Tag change, when an alarm goes active, and more!
- **Reports** - Use scripting to create a customized datasource, or create your own unique action to use with the scheduling system.
- **Alarm Notification** - Create custom rosters using scripting to dynamically change who gets notified with each new alarm event.
- **Client, Session, and Gateway** - Add a script that will run when certain events happen, such as when the Client or Gateway starts up, or on certain keystrokes.

Users that are new to Ignition focus mainly on the component binding system in [Vision](#) or [Perspective](#), and for good reason. It's simple, flexible, and generally easy to understand without much of a computer background. However, Ignition has a complete scripting system built into every place you can think of. Using it is not a requirement, but it can add a significant degree of flexibility and customization to your projects. It allows you to create exactly what you need, giving you total control where pre-canned options fall short.

The majority of your scripting will be done in Event Handlers inside of components. This system makes it very easy to get started scripting with little to no experience. With the script builders, basic scripting like Navigation and setting Tag values takes just a few clicks. See also [Script Builders in Vision](#) and [Component Events and Actions](#).

Related Topics ...

- [Getting Started with Scripting in Ignition](#)
- [Gateway Event Scripts](#)
- [Client Event Scripts](#)
- [Tag Event Scripts](#)
- [Scripting in the Report Module](#)

In This Section ...

Python Scripting

About Python

While it is entirely possible to create a complete and powerful project in Ignition without writing a line of script, many designers will find that in order to complete projects with specific requirements, they need to learn at least a little Python. In our experience, most industrial projects involve lots of very complex and specific requirements.

The good news is that learning [Python](#) is easy and [enjoyable](#). Python is one of the most beautiful programming languages we've ever encountered. It is very easy to read - even if you don't know it at all, you will probably be able to understand a basic Python script. It is frequently called "executable pseudocode". If you find yourself doing a lot of scripting, you may want to pick up a basic [reference book](#) about Python, or go through a simple [online tutorial course](#).

This section is a short tutorial specifically for Python, which should help get you started. It goes over all of the core concepts you will need for scripting in Ignition but then next section ([Scripting in Ignition](#)) goes over using Python directly inside Ignition.

- [Python Variables, Data Types, and Operators](#): Learn what a variable is and how to create it, what the various data types are, and what operators you can use on them.
- [Conditions and Loops](#): Learn the common if/else type of statements, as well as loops.
- Functions: Learn about the [built-in functions](#) that can do complex work with a simple command, as well as [user defined functions](#), which help compartmentalize the code.
- [Libraries](#): Learn about our built-in system functions, as well as pulling in outside libraries.

Python or Jython?

You'll often hear Python referred to as "Jython" by advanced users of Ignition. Python is the language, Jython is the implementation of the language that we use. Most users of Python use the implementation called "CPython" - they just don't realize it. See [http://en.wikipedia.org/wiki/Python_\(programming_language\)#Implementations](http://en.wikipedia.org/wiki/Python_(programming_language)#Implementations).

One of the powerful things about using Jython is that your script has access to the entire Java standard library. In the Client, this will be Java 6 or above. When running under the Gateway, this will be Java 8 and above. For more information, see [Accessing Java](#).

Many scripting users are blown away by their script's speed. We can't take credit for this - the Jython engine hot-compiles (compiles the code when it is run) your Jython code to Java bytecode, which means it runs natively in the JVM, which in turn can hot-compile it to machine code. It's fast.

Which Version of Python Is Used?

Ignition uses Jython 2.7. Jython is the Python programming language implemented over the Java Virtual Machine. When looking at outside documentation, such as on www.python.org, verify that you are looking at the correct version.

Jython 2.7 allows us to use the standard functions and tools in Python 2.7, so if you want to look up something in the Python docs, make sure to use version 2.7 (<https://docs.python.org/2/>).

Scripting Basics

Python is easy to learn, and with some understanding of its basic syntax, you can get started making your own scripts.

Hello World

Let's get right to everyone's favorite example, "Hello World." The following script will print out "Hello World" to the [Output Console](#).

The `print` keyword is a handy tool in Python, allowing you to write text to the Output Console. This is useful for debugging your scripts. You can print multiple things by separating them with commas.

Variables

Variables are created by simply assigning a value to them. Variables do not need to be declared, because Python has a dynamic type system. That means Python figures out the type of the variable on the fly when the script is executed.

The following script would print out: 15

On this page ...

- [About Python](#)
- [Python or Jython?](#)
 - [Which Version of Python Is Used?](#)
- [Scripting Basics](#)
 - [Hello World](#)
 - [Variables](#)
 - [Strings](#)
 - [Whitespace](#)
- [Comments](#)
 - [Individual Lines](#)
 - [Blocks of Lines](#)
 - [Comment Many Lines with a Keyboard Shortcut](#)
- [Control Flow](#)
 - [If and Else](#)
 - [For and While](#)



Basic Python - Variables and Comments

[Watch the Video](#)

Python - Using Variables

```
x=5  
y=3  
print x*y
```

Strings

Strings are defined in Python with a matching pair of 1 or 3 single or double quotes. There are few times when the type of quotation mark you use matters - but one common reason to choose one or the other is for 'escaping' other quotes inside your content. Some of the rules are shown here:

Python - Using Single and Double Quotes

```
print "This is my text"          # Using double quotation marks  
print 'This is my text'         # Using single quotation marks  
print "This is my text"         # This will not work because Python  
does not allow mixing the single and double quotation marks  
print "My name is 'David'"      # This will print: My name is 'David'  
print 'My name is "David"'       # This will print: My name is "David"  
print 'My name is Seamus O\'Malley' # This will print: My name is Seamus  
O'Malley
```

Triple quotes (single or double) can be used to make 'escaping' both single and double quotes inside your string easier, or to write multi-line comments:

Python - Multiple Lines of Comments Using a Triple Quote

```
'''  
This is a lot of text  
that you want to show as multiple lines of  
comments.  
Script written by Professor X.  
Jan 5, 1990  
'''  
print 'Hello world'
```

Strings can also be prefixed with certain characters to change how they are interpreted - for instance, a leading `u` character marks a string as Unicode, allowing for characters outside of the ASCII range to be used.

Python - Unicode Prefix on String

```
print u"ääöü"
```

Whitespace

Perhaps Python's most unique feature is logical blocks which are defined by an indentation in Python. A colon (`:`) starts a new block, and the next line must be indented (typically using a tab or 4 spaces). The block ends when the indentation level returns to the previous level. For example, the following will print out `"5 4 3 2 1 Blast-off!"` with each value on a new line. The final print is not part of the while loop because it isn't indented.

Python - Logical Blocks / Indentation

```
countdown = 5  
while countdown > 0:  
    print countdown  
    countdown = countdown - 1  
print "Blast-off!"
```

Comments

Comments are a way to document your Python script. There are several ways to use comments, but the best advice we can give is to use them as much as possible! There are a few ways to make a comment in Python.

Individual Lines

You can start a line with a pound/hash (#) sign, or put one anywhere in a normal line of code.

Python - Document Scripts Using Comments

```
# this is a comment
print 'Hello world' # this is also a valid comment
```

Blocks of Lines

While Python doesn't explicitly have a way to block comment (comment out multiple lines), [multi-line strings](#) are functionally similar, and a common convention.

Comment Many Lines with a Keyboard Shortcut

In Ignition, you can use the Ctrl-/ keyboard shortcut to comment several lines of code at once. Just highlight one or more lines of code and hold the Ctrl key and press "/". This will prepend all of the selected lines of code with the pound/hash (#) sign. Press Ctrl-/ again to remove the pound/hash sign.

Control Flow

Control Flow statements, that is the ifs and loops, make the language do things differently based on the various conditions. Python has all of the basic control flow statements that you'd expect from a programming language.

If and Else

An if statement allows you to check if a condition is true or not true. Depending on the condition, you can either execute a block of code, or do something else. Many of these can be chained together to determine under what conditions should certain code execute.

Pseudocode - If Statement

```
if condition == True:
    print value1
```



Basic Python - Flow Control

[Watch the Video](#)

For and While

Looping can be done with a for, which executes a block of code a set number of times, or a while, which executes a block of code until a certain condition is met. Both can be incredibly useful.

Pseudocode - For Statement

```
for item in myList:
    print item
```

Related Topics ...

- [Scripting in Ignition](#)
- [Getting Started with Scripting in Ignition](#)

[In This Section ...](#)

Variables, Data Types, and Objects

Without incorporating any Ignition-specific objects, it is important to understand the basics of Python. This section seeks to introduce three main Python principles: Variables, Data Types, and Operators. Additionally, this page and the sub-pages herein will provide plenty of examples to get you started.

This section of the manual attempts to introduce these principles so that they may later be used in conjunction with Ignition. The information in this section is far from comprehensive, as [Python's official documentation](#) is a better reference for all things Python. However, this page serves as a great way to jump right into the action.

Variables

Variables are created by simply assigning a value to them. Variables do not need to be declared, because Python has a dynamic type system. That means, Python figures out the type of the variable on the fly when the script is executed.

The following script would print out: 15

Python - Declaring and Assigning Variables

```
x=5  
y=3  
print x*y
```

On this page ...

- [Variables](#)
- [Built-in Data Types](#)
 - [None](#)
 - [Booleans](#)
 - [Strings](#)
 - [Numeric Types](#)
 - [Colors](#)
 - [Lists and Tuples](#)
 - [Dictionaries](#)
 - [JSON](#)
 - [Datasets](#)
 - [Dates](#)
- [Using the In-Keyword](#)
- [Basic Operators](#)
 - [Operator Reference](#)
 - [Order of Operations](#)

A space may be included on either side of the assignment operator ('='), but is not required. Thus, the following example would be functionality identical as the example above.

Python - Declaring and Assigning Variables with Spaces

```
x = 5  
y = 3  
print x * y
```



Python Variables

[Watch the Video](#)

Built-in Data Types

Python features several built-in data types. Below is an overview and links to appropriate pages where applicable.

None

There is a special value in Python called `None` (with a capital N). This is simply a special value that means no value. This value is equivalent to Java's `null` value. `None` can be used to initialize a variable, but is best when checking to see if something exists before doing extra work:

Python - None

```
# If some value is not equal to None...  
if something != None:  
    # ...then do some work  
    doWork()
```



Python Datatypes

[Watch the Video](#)

Booleans



Python has two built-in values to represent true and false values: `True` and `False`, respectively (note the capital letters). These can be used when testing for truth, and are implicitly returned when the comparison operator is used:

Python - Booleans

```
# Prints True
print 1 == 1

# Prints False
print 1 == 0
```

When using Booleans, such as predicates in an if-statement, you typically don't have to use `True` or `False` directly. Instead, many other values are considered True or False. The following values are also considered False:

- None
- Numeric values of 0, such as 0
- Empty Sequences and Dictionaries, like [] or {}

Values aside from the ones mentioned above are considered true, so you can easily utilize the existence of a non-zero value as a True. Note that for integers both positive and negative numbers are True, only a value of 0 is False.

Strings

Literal `strings` can be typed in using either double quotes or single quotes. This can be handy when your string contains one quote or the other. You can also use the backslash character to escape special characters including these quotes. See the [Strings](#) page for more information.

Numeric Types

Numbers can just be typed in normally, like `42` or `3.14159`. Adding a decimal point differentiates an Integer from a Float. More information on Numeric types can be found on the [Numeric Types](#) page.

Colors

Working with colors in Python is remarkably easy. You can simply use any tuple of 3 or 4 integers to represent a color in RGB or RGBA. For example, to set a label's text color to red, you can simple do something like this:

Python - Tuple as Color

```
label = event.source
label.foreground = (255,0,0) #(red,green,blue)
```

Additionally, the `system.gui.color` function allows you to pick a color in a similar fashion:

Python - System Function as Color

```
newColor = system.gui.color(255,0,0)
```

Lists and Tuples

Python offers a variety of sequence types: most notably [Lists and Tuples](#). These are ordered collections, meaning they are indexed and the sorted order is maintained. More information on these types can be found on the [Lists and Tuples](#) page.

Python - System Function as Color

```
 newList = [1,2]
newTuple = (2,5,7)
```

Working with Different Datatypes

[Watch the Video](#)

Dictionaries

A [Dictionary](#) is a very useful type that holds a set of key-value pairs. Unlike sequences, they are not ordered, so there is no need to sort them. Instead you give each Value in the dictionary a Key, which handles as a reference to Value. You may have used these in other languages and know them as hashmaps, maps, associative memories, or associative arrays. More information on Dictionaries can be found on the [Dictionaries](#) page.

Python - System Function as Color

```
newDictionary = {"itemName": 5}
```

JSON

JSON stands for JavaScript Object Notation. While it is not a data type, it is a way of defining data in a human readable format, and is commonly used in many applications. It comes from the Javascript programming language, but it is language independent. Each JSON object contains lists and objects. A [list](#) is a series of ordered values separated by commas that is commonly used within Python. The object works like a [dictionary](#), using any number of name/value pairs, where each value could be any basic data type, a list, or even another object with its own name/value pairs. Because JSON is just a way of defining data, it can be used in many different programming languages, including Python. This makes it a useful tool for defining data in a way that humans can easily read.

Ignition also has two scripting functions that allow you to convert between a JSON string and a native Python object: [system.util.jsonEncode](#) and [system.util.jsonDecode](#).

Official Documentation

For more information on JSON, see <http://www.json.org>.

Datasets

A [Dataset](#) is a multidimensional collection of values, stored in a manner similar to how values on a spreadsheet appear. Python does **not** natively have a Dataset type. Instead these datasets were created for use inside of Ignition. There are two types of datasets:

- **Dataset:** Sometimes called a "Standard Dataset", this type is commonly used on many Vision Components, such as the [Power Table](#) or [Chart](#), to display multiple values simultaneously.
- **PyDataset:** Short for "Python Dataset", these datasets act in a manner very similar to a Python Sequence when it comes to accessing specific values, or iteration (see [Lists and Tuples](#)). Ignition's built-in system functions that interact with the database typically return a PyDataset.

While there are two types of datasets, you can easily convert one type of dataset to the other. Additionally, you can easily create a dataset from a script.

Python - Creating a Dataset

```
header = ["The Only Column"]
rows = [[1],[2],[3]]

myDataset = system.dataset.toDataSet(header, rows)
```

Dates

[Dates and times](#) can be created in Python with the `datetime` and `time` libraries. However, the Ignition's built-in `system.date` functions can also be used instead without having to import either library.

Python - Show the Current Datetime

```
currentTime = system.date.now()

print "The current time is: %s" % currentTime
```

Using the In-Keyword

Python's `in` keyword can be used to check the contents of something for a specific instance of another object. One use is to check the contents of a string for a certain substring:

Python - Using In Keyword to Look for a Substring

```
myString = "Hello World"
subString = "World"

# Here we are using an if-statement to look for the substring. If the substring exists inside of myString,
# the expression returns True.
if subString in myString:
    # This will print out, because the substring "World" exists in the string "Hello World"
    print "The word 'World' appears in myString"
```

The `in` keyword can be used to look for a certain object inside of a sequence:

Python - Check for the Presence of an Object

```
myList = [1,2,3,4]

# If the integer 3 exists in myList, then this will print True, otherwise it will print False.
print 3 in myList
```

Python - Check for the Presence of an Object

```
# Check for the Administrator role.
if "Administrator" in system.security.getRoles():
    print "Administrator found"
```

Additionally, the `in` keyword can be used in an expression on a `while` loop:

Python - Check for the Presence of a Keyword in an Expression

```
myList = [1,2,3,4]

# As long as the integer 4 is in myList, the while loop will continue to iterate.
while 4 in myList:
    # pop(0) will remove the first element in the list
    myList.pop(0)
else:
    print "all done!"
```

Basic Operators

Python has many common operators as you would expect, or are at least familiar with if you've worked with other scripting languages.

These are just the basics. There are other operators, like bit shift operators and more. Read about them at: <http://docs.python.org/library/stdtypes.html>

Operator Reference

Arithmetic Operators



In regards to Arithmetic Operators, the precision of the returned value depends on the data types of the arguments: including at least one `float` will return a `float`, otherwise, an `integer` is returned.

Operator	Meaning	Example	Output
+	Addition. Note that the data type of the returned object depends on the data type of the arguments being used:		14

		<pre>print 5 + 9 print 5.0 + 9</pre>	14.0
-	Subtraction	<pre>print 5 - 9</pre>	-4
*	Multiplication	<pre>print 5 * 9</pre>	45
/	Division	<pre>print 10 / 4.5</pre>	2.22222222222 22223
//	Floored Division: will divide and return just the nearest integer value, even if dividing floats.	<pre>print 10 // 4.5</pre>	2.0
%	Modulo: returns just the remainder of the dividend (the left argument) after being divided by the right argument. In the example on the right, the 4.5 divides evenly twice, leaving 1.0 as a remainder.	<pre>print 10 % 4.5</pre>	1.0
**	Power: raise the number on the left to the power of the number on the right.	<pre>print 5 ** 3</pre>	125

Boolean Operators

Operator	Meaning	Example	Output
or	Returns True if either argument is True. If both are False, then returns False.	<pre>x = False y = True print x or y</pre>	True
and	Returns True only if both arguments are True. Otherwise, returns False.	<pre>x = False y = True print x and y</pre>	False
not	Returns a boolean value that represents the opposite value of the trailing expression: False becomes True, and True becomes False.	<pre>x = True print not x</pre>	False

Comparison Operators

Operator	Meaning	Example	Output
<	Less than	<pre>print 10 < 5 print 10 < 10</pre>	False False

<code><=</code>	Less than or equal to	<pre>print 10 <= 5 print 10 <= 10</pre>	False True
<code>></code>	Greater than	<pre>print 10 > 5 print 10 > 10</pre>	True False
<code>>=</code>	Greater than or equal to	<pre>print 10 >= 5 print 10 >= 10</pre>	True True
<code>==</code>	Equal	<pre>print 10 == 10 print 10 == 0</pre>	True False
<code>!=</code>	Not equal	<pre>print 10 != = 10 print 10 != = 0</pre>	False True
<code>is</code>	Returns True if both arguments are referring to the same object, otherwise, returns False.	<pre>x = 10 y = 5 print x is y</pre>	False
<code>is not</code>	Returns True if both arguments are referring to different objects.	<pre>x = 10 y = 5 print x is not y</pre>	True

Order of Operations

Note that there is an order of operations for Arithmetic and Boolean operators that can be modified with parenthesis. For arithmetic they are grouped in tiers and evaluated left-to-right within a given tier:

- Parenthesis
- Exponents
- Multiplication, Division, Modulo
- Addition, Subtraction

Python - System Function as Color

```
6+10*2    # produces 6+(10*2) = 26
6*10+2    # produces (6*10)+2 = 62

10*6%2    # produces (10*6)%2 = 0
10%6*2    # produces (10%6)*2 = 8
```

For Boolean operators:

- Not
- And
- Or

Related Topics ...

- [Conditions and Loops](#)

In This Section ...

Numeric Types

The Differences Between the Types

Within Python, there are a few numeric types that we will often use within the scope of Ignition: integers, floats and booleans. **Integers** have at least 32 bits of precision and consist of non decimal values (1, 2, 50, 246). When working with larger numbers, Python also supports a **long** type that has unlimited precision. **Floats** consist of numbers with decimal values (1.0, 2.4, 50.7, 246.8734). **Booleans** are a unique subset of integers. They are either **True** or **False**, but can also be represented using a 1 or 0, respectively.

On this page ...

- The Differences Between the Types
 - Numbers in Arithmetic
 - Operators and Functions

Numbers in Arithmetic

When using numeric values in expressions or equations, Python is very flexible in how it handles multiple types in the same operation. When using two of the same type in an expression, the outcome will always be of that type. Adding two integers together yields an integer, while multiplying two floats will yield a float. With division, using integers will always result in a floor division so that the answer remains an integer. Floor division will simply remove the decimal, and will not round up at any point.

When using different types, the more precise type will be used for the outcome. Doing an operation with one integer and one float will produce an outcome that is a float. Adding a float and an integer together will yield a float.

Below, we have some examples that show off some of the different ways that Python handles expressions. You can try these out in the interactive interpreter of the [Script Console](#).

Type of Arithmetic	Description	Example	Output
Addition and Subtraction	Adding/Subtracting integers will return an Integer. If at least one of the numbers is a float, then the resulting number will also be a float. Otherwise the result will be an integer	<pre>4 + 5 # int + int = int 4.0 + 5 # float + int = float 4.0 + 5.3 # float + float = float</pre>	9 9.0 9.3
Multiplication	Similar to Addition and Subtraction, multiplication will only return a float if at least one of the numbers is a float	<pre>4 * 5 # int * int = int 4 * 5.0 # int * float = float</pre>	20 20.0
Division	Dividing an integer by another integer results in floor division: meaning the remainder of division is never returned. Note that this is not the same as rounding to the nearest integer. To include the remainder, at least one of the numbers needs to be a float	<pre>4 / 5 # int / int = int 4 / 5.0 # int / float = float</pre>	0 0.8

Operators and Functions

Python offers a wide range of operators that can be used to perform calculations on numeric values. Additionally, there are many functions that can perform specific operations to numeric values. The tables below contain some of the most common operations and functions, but this is not an exhaustive list.

Operators			
Operator	Description	Example	Types that use the Operator
+	Used to find the sum of two numeric values Can also be used to concatenate string values	<pre>5 + 1 = 6 "Hello " + "World"</pre>	Numeric Types, Strings

-	Used to find the difference of two values	5 - 1 = 4	Numeric Types
*	Used to find the product of two values	5 * 2 = 10	Numeric Types
/	Used to find the quotient of two values	4 / 2 = 2	Numeric Types
%	Modulus operator. When used with numeric types, returns the remainder of the quotient of two values The operator is also seen in string formatting .	5 % 2 = 1	Numeric Types, Strings
**	Used to find a value to the power of another value	5 ** 2 = 25	Numeric Types

Functions

Function	Description	Example
int(x)	Converts x to an integer	int(4.8) = 4
long(x)	Converts x to a long	long(5.7) = 5
float(x)	Converts x to a float	float(4) = 4.0
abs(x)	Absolute value of x	abs(-4.6)=4.6
round(x[, n])	Rounds x to n digits. If n is not specified, default is 0	round(5.674)=6.0 round(5.674, 2)=5.67

Related Topics ...

- [Strings](#)
- [Lists and Tuples](#)
- [Dictionaries](#)
- [Datasets](#)
- [Dates](#)

Strings

What are Strings?

In Python, as well as most other programming languages, a string is a grouping or string of characters that are used literally instead of as names of variables or other objects. In Python, they can be enclosed in either single quotes or double quotes:

Python - Print Statements

```
# These two print statements will both print out a sentence that looks the same as the other.  
print 'These will both print the same!'  
  
print "These will both print the same!"
```

Strings are also considered sequences, which means they can be [iterated through](#) much like a list.

On this page ...

- [What are Strings?](#)
 - [String Escape Character](#)
 - [Multi-Line Strings](#)
 - [Raw Strings](#)
 - [Unicode Strings](#)
- [Combining Strings](#)
- [String Indexing](#)
 - [Slicing Strings](#)
- [String Formatting](#)
- [String Search Example](#)
- [String Functions](#)

String Escape Character

Strings use a special character called an "escape" character to denote when something different should be happening within the string. In Python, the escape character is the backslash (\). The escape character is mainly used in a few different ways:

The Escape Character and Quotation Marks - "" and ''	Comments	Example	Output
	Here, we use the escape character to let Python know we want to ignore the second single quote and directly print it. This works with either single or double quotes.	print 'doesn\'t'	doesn't
	Alternately, we can enclose the string in the opposite quotes that we are using in the string. Because we enclose the string in double quotes here, the single quote prints normally.	print "doesn't"	doesn't
	This also works in reverse, where we can enclose the string in single quotes and use double quotes within.	print '"Test", he said.'	"Test", he said.
	Of course, we can also use the escape character with double quotes enclosing the string.	print "\"Test\\\"", he said."	"Test", he said.
New Lines and Tab Spacing			
	A new line may be specified with the escape character and "n". Alternatively, a multi-line string may be used to add line breaks to a string literal.	print "First line\nSecond line"	First line Second line
	Tab-spacing may be added to a string literal with the escape character and "t"	print "Hello \tWorld"	Hello World
Specifying a Backslash			

If you need to place a backslash inside of a string literal, simply use the escape character twice.

```
print "Folder\\file.txt"
```

```
Folder\file.txt
```

Multi-Line Strings

If a string literal should contain multiple lines, you can use a multi-line string, which is created by using three quotation marks.

Example	Output
<pre>print '''This is on multiple lines'''</pre>	This is on multiple lines

This is useful when using the `system.db` functions, as full SQL queries are typically passed to these functions, so using a multi-line string allows you to write the query in an easy to read format.

Python - Multi-line String

```
query = '''      SELECT * FROM myTable
                  WHERE id < 10'''

system.db.runPrepQuery(query, [])
```

Raw Strings

Sometimes, it is necessary to print the raw string without allowing escaped characters. This is done by placing the letter "r" in front of the string.

Python - Raw String

```
print r'This string \n will print out directly as written.'
```

This is especially useful in cases where a file path needs to be hard coded as a string literal:

Python - File Path as a Raw String

```
# Specifying a Windows file path
myPath = r"C:\Folder\Another Folder\file.txt"
print myPath

# Specifying a Linux file path
myPath = r"/home/Directory/Another Directory/file.txt"
print myPath
```

Unicode Strings

Strings that contain characters beyond 7-bit ASCII, such as é or ? need to be marked as unicode strings by placing the letter u in front of the string.

Python - Unicode String

```
print u'été'
```

Combining Strings

Two different string type variables can actually be combined or concatenated using the plus (+) sign. It is important to understand that they are concatenated exactly.

Python - Concatenated String Type Variables

```
a = "this"  
b = "that"  
  
# Will print 'thisthat' because there was no space at the end of a or the beginning of b.  
print a + b  
  
# Will print 'this that' because we added a space between them.  
print a + " " + b
```

String Indexing

Strings in Python are actually indexed, with the first character having an index of 0. To grab a value at a specific index, you place a value within square brackets ([]) after the string variable.

Python - Indexed Strings

```
a = "Ignition"  
  
# Will print out 'I', since it is in the zero position.  
print a[0]  
  
# Will print out 't', since it is in the fourth position.  
print a[4]
```

You can also use a negative index, which will start from the right side of the string with the last character having a index value of -1. This is useful for getting the last character when you aren't sure how long the string is.

Python - Negative Indexed String

```
a = "Ignition"  
  
# Will print out 'o', since it is the second to last character.  
print a[-2]
```

Slicing Strings

Using the string index values, we can actually grab slices or parts of the string. Similar to grabbing an individual character, we place two values separated by a colon within square brackets. You can think of these numbers as the slices between characters, with 0 before the first character, 1 before the second, and so on. It should look like this: string[4:7] The first value is the start location, while the second value is the end location.

Python - Sliced Strings

```
a = "Inductive Automation"  
  
# Will print out 'ctive Auto'. Note how the space is counted as a character,  
# and the 14th location is before the m, which is not included.  
print a[4:14]
```

If left blank, the first value will default to 0, while the second value will default to the length of the string. Additionally, negative values can be used as well, just like with the index.

Python - Negative Value Sliced Strings

```

a = "Inductive Automation"

# From index 0 to index 12.
print a[:12] # Will print out: Inductive Au

# From index 12 to the end of the string.
print a[12:] # Will print out: tomation

# From the 4th character from the end to the 5th index. Note this prints nothing because the start character
# is after the end character.
print a[-4:5] # Will print out:

# From the 8th character from the end to the 2nd character from the end.
print a[-8:-2] # Will print out: tomati

# From index 5 to the 5th character from the end.
print a[5:-5] # Will print out: tive Autom

```

You can use all of these together (and with integer variables or functions) to have a very flexible way to format strings.

String Formatting

Also known as string substitution, it allows you to enter in values into a string, similar to having a variable inside of the string. This is useful when doing a database query where you can build the query as a string, and then add in the query parameters when executing the query.

Doing this requires the use of the percent (%) sign followed by the parameter type within the string. After the string, you then use the percent sign followed by the parameter values within parentheses in order.

Python - String Substitution

```

a = "I have %i apples."

# Will print exactly as written, because we are not substituting any values in.
print a

# Will print with an 8 substituted in for the %i. I have 8 apples.
print a % (8)

```

Instead of using literal values, you can also use variables as parameter values instead. This is useful when pulling the value from a function. Each string can also have multiple values substituted, as long as the parameters are in the order that they are in the string.

Python - String Substitution using Variables as Parameters

```

apples = 10
oranges = 14
peaches = 5

a = "I have %i apples, %i oranges, and %i peaches."

# Will print with the variable values substituted in for the %i. I have 10 apples, 14 oranges, and 5 peaches.
print a % (apples, oranges, peaches)

```

There are a few different types used with string formatting, the most common of which are listed here. The character values are what go after the percent sign.

Type	Character
Signed Integer	i
Floating Point Decimal Format	f
Floating Point Exponential Format	e
String	s

String Search Example

You can bring all this together for a simple way to print out just the last word in a string.

Python - Find the Last Word of a String

```
# Find just the last word of a string.  
myString = "Inductive Automation"  
  
lastSpaceIndex = myString.rfind(" ")  
  
# Add one to not include the space.  
print myString[lastSpaceIndex+1:] # Will print out: Automation
```

String Functions

Python strings have many functions available that can manipulate the string or give information on the string. The most common are in the table below.

Function	Description	Example	Output
len(string)	Returns the length of the string .	a = "Inductive Automation" print len(a)	20
x in string	Will return True if x is within the string, False if not. Can also be used to iterate through the string.	string = "Inductive Automation" if 'd' in string: print "There is a d in the string" for letter in string: print letter	There is a d in the string I n d u c t i ...
string.find(x, [start[, end]])	Returns the first location of the substring x from the string. Returns -1 if the substring is not found.	string = "Inductive Automation" print string. find(" ")	9
string.rfind(x, [start[, end]])	Similar to find, but returns the last location of the substring x . Returns -1 if not found.	string = "Inductive Automation" print string. find("i")	17

string.upper()	Returns a copy of the string with all characters uppercase. This is useful when comparing user input to a string value, as the user may use a different case for certain letters.	<pre>userInput = "ADMINISTRATION" print userInput.upper()</pre>	ADMINISTRATION
string.lower()	Returns a copy of the string with all characters lowercase. This is useful when comparing user input to a string value, as the user may use a different case for certain letters.	<pre>userInput = "ADMinISTRAtion" print userInput.lower()</pre>	administration
string.capitalize()	Returns a copy of the string with the first character capitalized and all other characters lowercase.	<pre>string = "here is my sentence." print string.capitalize()</pre>	Here is my sentence.
string.title()	Returns a copy of the string with the first letter of each word capitalized and all other characters lowercase.	<pre>string = "here is my sentence." print string.title()</pre>	Here Is My Sentence.
string.strip([x])	Returns a copy of the string with leading and trailing characters removed, where x is the string of characters.. If x is omitted, then removes whitespace from the leading and trailing edges of the string. lstrip() and rstrip() may instead be used to strip characters from the leading or trailing edge	<pre>string = " This string has some empty space" print string print string.strip() fencedString = " __My String__ _ " print fencedString.strip("_ ") print fencedString.lstrip("_ ") print fencedString.rstrip("_ ")</pre>	This string has some empty space This string has some empty space My String My String_____ _____ _My String
string.count(x[, start[, end]])	Returns the number of occurrences of x in the string. A start and end can be specified that will limit the count to that area.		2

	end]])	<pre>string = Inductive Automation"</pre> <pre>print string. count('i') print string. count('i', 4, 9)</pre>	1
string.split ([delimiter[, maxsplit]])	Returns a list of the words in the string. Optionally, specifying a delimiter will split the string on the delimiter string. Specifying a maxsplit will split the string a maximum number of times, with the remainder of the string as the final list object. The number of items in the list will not be more than maxsplit + 1	<pre>sentence = "This is an example of split"</pre> <pre>print sentence. split()</pre> <pre>dashedSentence = "Yet- another- sentence- here"</pre> <pre>print dashedSentence .split("-", 2)</pre>	['This', 'is', 'an', 'example', 'of', 'split'] ['Yet', 'another', 'sentence- here']
string.rsplit ([delimiter[, maxsplit]])	Similar to split, but splitting is performed from right to left. Thus if maxsplit is smaller than the total number of delimiter in the string, only the rightmost words will be split off as separate items.	<pre>sentence = "This is an example of split"</pre> <pre>print sentence. rsplit()</pre> <pre>dashedSentence = "Yet- another- sentence- here"</pre> <pre>print dashedSentence .rsplit("-", 2)</pre>	['This', 'is', 'an', 'example', 'of', 'split'] ['Yet- another', 'sentence', 'here']
string.join(x)	Returns a copy of the string that is the concatenation of the strings in the iterable x . The string calling the join is what will separate the values of the iterable.	<pre>a = '-' b = 'abcdef'</pre> <pre>print a.join (b)</pre>	a-b-c-d-e-f
string.replace(old, new[, count])	Returns a copy of the string where occurrences of the old substring are replaced with the new substring. Optionally, if count is specified, will only replace the first count number of occurrences.	<pre>string = Ignition is good!"</pre> <pre>print string.</pre>	Ignition is awesome!

```
replace  
("good",  
"awesome")
```

Related Topics ...

- [Numeric Types](#)
- [Lists and Tuples](#)
- [Dictionaries](#)
- [Datasets](#)
- [Dates](#)

Lists and Tuples

Sequences

Like [Strings](#), Python has two other common sequence types: Lists and Tuples. While lists and tuples are similar to strings in that they share many of the same functionality, they are unique in that they are used to group other values together. Both lists and tuples define a number of values separated by commas, but lists are enclosed in square brackets [] and are mutable, meaning their contents can change, while tuples are enclosed in parentheses () and are immutable, meaning their contents can't change.

Lists

As stated above, lists are groups of comma separated values enclosed in square brackets, but can utilize many of the features available to other sequences like strings.

Python - Lists

```
# Lists are very simple to create.  
myList = [1, 2, 3]  
  
print myList # Will print out: [1, 2, 3]  
  
# Empty lists can also be created, to have items added to them later.  
myList = []  
  
# Lists are not confined to hold values of a single data type either.  
myList = [1, "hello", 3.3]  
  
# Lists can even hold other lists! In this case, myList would actually  
hold 5 elements:  
# 1, a list, 4, 'that', and another list. The last list also contains a  
list as well.  
myList = [1, ["this", 3.3], 4, 'that', [6, [7.7, 'other'], 9]]
```

On this page ...

- [Sequences](#)
- [Lists](#)
 - [List Concatenation](#)
 - [List Indexing](#)
 - [Appending to Lists](#)
- [List Functions](#)
- [Tuples](#)
- [Tuple Functions](#)



INDUCTIVE
UNIVERSITY

Basic Python - Lists and Dictionaries

[Watch the Video](#)

List Concatenation

Like strings, lists also support concatenation and can be combined to form a single larger list.

Python - List Concatenation

```
a = [1, 2, 3]  
b = ['four', 'five', 'six']  
  
print a + b # Will print out [1, 2, 3, 'four', 'five', 'six']
```

List Indexing

Like strings, lists are also indexed, which allows you to grab single values or splice them to get ranges of values. Just like everything else in Python, lists start with 0.

Python - List Indexing

```
myList = ['a', 'b', 'c', 'd', 'e', 'f', 'g']  
  
print myList[3] # Will print out: d  
  
print myList[-2] # Will print out: f  
  
print myList[2:5] # Will print out: ['c', 'd', 'e']
```

When trying to index a nested list, we simply need to add a second index after the first.