TYPE and WHERE Parameters

Most webjects supported by the Windchill adapter accept parameters named TYPE and WHERE. These parameters combine to specify database query criteria:

- The TYPE parameter identifies the type of business object to query.

  An acceptable value of the TYPE parameter is the name of a modeled Windchill business object class or a subtype defined in the Windchill Type and Attribute Management utility. For more information, see Valid Syntax for TYPE Parameters.

- The WHERE parameter identifies the attribute values and query criteria to be matched.

  An acceptable value of the WHERE parameter is a query expression. A valid query expression contains one ore more query terms. For more information, see Valid Syntax for WHERE Parameters.

Valid Syntax for TYPE Parameters

Use the internal name of the type in the TYPE parameter. The internal name can also be used in custom Info*Engine tasks and other code where there is a need to refer to a type.

You can set the internal name of a type when loading out-of-the box types or when you create (or update) a type through the Type and Attribute Management utility.

The following rules apply to the internal name:

- It must be unique across all types in the system.
- You are limited to 50 characters.
- Only letters, numbers, periods, and underscores are supported.

The internal name is typically the same as the type name. If you are unsure of whether a type has an internal name, you can view the information from the Type and Attribute Management utility.

Tip

If the type does not have a internal name, you can add one by updating the type using the Type and Attribute Management utility. For details on accessing the Type and Attribute Management utility, see the *PTC Windchill Basic Administration Guide*.

Valid Syntax for WHERE Parameters

An acceptable value of the WHERE parameter is a query expression. A valid query expression contains one ore more query terms:

- A *term* specifies an attribute name, a relational operator, and a value or wildcard pattern.
- A *negation operator* might preface a term to request that an inverse database matching operation be applied.
- Terms can be connected by boolean operators to produce conjunctive (AND) and disjunctive (OR) expressions.
- Sub-expressions can be enclosed by parenthesis to control order of execution.

The valid syntax for WHERE parameters is as follows:

- Quoted strings define terminal symbols (literal character sequences that must occur in a valid WHERE value).
- A name enclosed by "<" and ">" is a non-terminal symbol.
- A sequence enclosed by "(" and ")*" indicates that zero or more occurrences are accepted.
- Alternative productions for a non-terminal are delimited by the vertical line character "|".

```
<expression> ::= <subexpr>
                 | "()"
   <subexpr> ::= <term> (<boolean> <subexpr>)*
      <term> ::= <name> <relational> <value>
                 | "!" <term>
                 | "(" <expression> ")"
   <boolean> ::= "&" | "|"
<relational> ::= "=" | "!=" | "<" | "<=" | ">" | ">="
      <name> ::= attribute-name
     <value> ::= <quoted> | <unquoted>
    <quoted> ::= "' " <unquoted-value> "' "
  <unquoted> ::= <exact> | <wildcard>
<wildcard> ::= <wild_char> (<wildcard>)*
 <wild_char> ::= "*" | character
```

Where:

- *attribute-name*—The internal name for any attribute of the business object type specified by the TYPE parameter.
  - The internal name of a hard (modeled) attribute is simply the name of the attribute.
  - The internal name of a soft attribute is assigned in the Type and Attribute Management utility.

  These internal names can be overridden in **<*Windchill*>/codebase/LogicalAttributes.xml**.
- *"()"*—The special expression "()" generates a query that returns all business objects of the type specified by the TYPE parameter.

  This should be used with caution as it might return a very large number of business objects. Other expressions generate queries that return the business object, if any, that match the indicated criteria. In addition, webjects accept multiple WHERE parameter values. In this case, the multiple values are concatenated with the "&" operator to form the query expression that will be executed.
- *boolean*—The boolean operator "&" is the AND operator, and the boolean operator "|" is the OR operator.

  When both are specified in an expression, "&" has higher precedence than "|", so the sub-expressions connected by "&" are executed first. However, parenthesized expressions are always executed before any other part of the expression as a whole. Therefore parenthesis can be used to override the default precedence rules.

  Nested parenthesized expressions are executed from the deepest to the most shallow. If using a JSP page, an AND can be represented in the WHERE clause as "&." If using XML tasks, the AND must be specified using the HTML encoded value for "&," which is "&."
- *value*—An acceptable attribute value is any value or wildcard string that is valid for the attribute.

  Note

Wildcards only work for those attributes whose type is a string. When pre-processing a `where` clause, the `value` portion of a `name=value` pair is translated to the type for the corresponding attribute. As a result, wildcards only work for string attributes.

For example, assume you have an attribute named “count” and it is an integer. A `where` clause containing “count=1” is valid because 1 can be translated into an integer. However, “count=1*” is not valid because “1*” cannot be translated into an integer.

If you would like to specify numeric ranges, use the > < operators.

Specifying Case with the WHERE Parameter

When using the WHERE parameter, searches are insensitive to case by default. To make a search case sensitive, specify the WHERE_CASE_SENSITIVITY with a value of TRUE. The WHERE_CASE_SENSITIVITY parameter can be specified for all webjects which take a WHERE parameter.

WHERE and TYPE Examples

The following examples illustrate WHERE parameter values that might be specified when the TYPE parameter is specified as `wt.part.WTPart`:

Return all parts	`()`
Find a part named “engine”	`name=engine`
Find all parts whose names contain the substring “engine”	`name=*engine*`
Find all parts whose names contain the substring “en” followed by any sequence of characters followed by “ine” ending with any sequence of characters	`name=*en*ine*`
Find parts whose names begin with “engine” or end with “body”	`name=engine*|name=*body`
Two ways to find parts whose names end with “body” and do not contain “engine”	`name=*body&!name=*engine*` `name=*body&name!=*engine*`
Find parts whose names begin with “engine” or end with “body” and do not have “9” in their part numbers	`(name=engine*|name=*body)&number!=*9*`