pjs.define()

The pjs.define() API is used within Profound.js modules to declare fields, arrays, and data structures with a strong data type.

To declare fields that are loosely typed, use the var clause in JavaScript instead.

Parameters

1. Field name (String)
2. Config (Object)

Shorthand Parameters

1. Field name (String)
2. Type (String)
3. Length (Number) - specify only if required for particular Type
4. Decimals (Number) - specify only if required for particular Type

Config object

A config object is required when defining fields. It is what gives your field attributes, like type, length, and decimal places. The following config properties are available:

Property name	Property type	Description
type	String	The following data types can be specified: • "packed decimal" or "packed" (both variations have the same meaning) • "decimal" or "zoned" or "zoned decimal" or "numeric" (all variations have the same meaning) • "integer" • "unsigned integer" • "char" • "varchar" • "boolean" • "date" • "time" • "timestamp" • "float" or "number" (both variations have the same meaning) • "pointer" • "string" • "data structure" or "ds" (both variations have the same meaning) Some config properties only apply to certain types.
length	Number	Specifies the field length. It is required for most data types; exceptions are: • date • time • timestamp • pointer • data structure
decimals	Number	Specifies the number of decimal positions. It required for packed and zoned fields.
varying	Boolean	Specifies that a character field is of varying length; otherwise, the field will be a fixed length field padded with spaces.
initValue	String /Number /Boolean /Array	Specified the field's initial value.
dim	Number	This property turns the definition into an array. When declared this way, arrays start with an index of 1, similar to RPG.
orderBy	String	Specifies the default sort order for arrays declared with the dim property. The orderBy property is used by array-related API, such as pjs.lookup(). Possible values are: • "ASCEND" • "DESCEND"

elements	Object	Specifies an object of subfields for a data structure. Each property within the object is the subfield name, while the property value is another config object.
qualified	Boolean	Identifies a data structure as qualified, meaning that all references to its subfields must be qualified with the data structure name first; for example: MyDS.mysubfield = "myvalue". This makes the data structure similar to a JavaScript object. If qualified is not specified, the subfields can be referenced directly; for example: mysubfield = "myvalue".
special	String	Identifies a special subfield and can only be specified on a data structure subfield configuration with the elements property. Possible values are: For a Status Data Structure (statusDS): • "*status" - Status Code (commonly used to identify errors) • "*proc" - Module Name • "*parms" - Number of parameters passed • "*routine" - Name of the routine in which an exception or an error occurred For an Information Data Structure (infDS): • "*file" - The first 8 characters of a file name • "*record" - The first 8 characters of a record format name being processed • "*opcode" - Operation • "*status" - Status Code • "*routine" - The first 8 characters of the name of a routine in which a file operation was done
nullable	Boolean / String	Use this property to create null-capable fields. • Use true to just make the field null-capable. This option allows you to assign a null value to the field. • Use a string to specify a reference a Boolean field to use as the null-indicator and to make the field null-capable.

extName	String / Object	Specifies the name of a file that contains the fields used as the subfield description for the data structure being defined. The property can specified as a String that contains the file name or as an object with the following properties:
		
		• file - name of the file
		• format (optional) - name of the format within the file; if not specified, the first record format is used
		• extract (optional) - specifies which fields to extract using one of the following values:
		  • "*all"
		  • "*input"
		  • "*output"
		  • "*key"
		  • "*null"
		
		The file name can be qualified with a library; for example: "PRODLIB/MYFILE".

like	String	Name of an already existing field which this field should base its type on.
likeRec	String / Object	Specified on a data structure to make its definition like a previously defined database table. The property can be specified as a String, which names the previously defined record format, or as an object with the following properties: • record - internal record format name from a previously defined table, specified as a String. • fields - optional String value of "*key". If "*key" is specified, only key fields will be defined. When likeRec is specified, the data structure becomes qualified automatically.
likeDS	String	Used on a data structure to specify the name of another data structure from which the configuration is derived.
dataArea	String	Specifies a data area object to be associated with the declared field. The name of the data area may include a library. For example: • "MYDTAARA" (data area will be searched for using the pathlist / IBM i library list because the library name was not specified) • "PRODLIB/MYDTAATA" (the library name is explicitly specified here) The data are object name can also be dynamic based on a Profound.js field. This is done by specifying a JavaScript object for the property that looks like this: • { field: "FieldName" }
statusDS	Boolean	Identifies a Status Data Structure. A Status Data Structure is automatically populated with information about the current Profound.js module and job. It is an easy way to retrieve information, such as the current IBM i job name, the current user id, and more.
userDefinedData	Any	Specifies user-defined general purpose data associated with the field configuration.
based	String	The name of a pointer field that the declared field will be based on. The pointer will be defined automatically, if necessary.

dateFormat	String	The format for a date type field. The available date formats are: • "*job" • "*mdy" • "*dmy" • "*ymd" • "*jul" • "*iso" • "*usa" • "*eur" • "*jis" This property is ignored if the field is not a date field.
dateSeparator	String	The separator for a date type field. The available separators are: ▪ '-' (hyphen) ▪ '.' (dot) ▪ ',' (comma) ▪ '&' (ampersand) ▪ ' ' (space) ▪ '/' (forward slash) This property is ignored if the field is not a date field, and also ignored for date fields with format "*iso", "*usa", "*eur", and "*jis".
timeFormat	String	The format for a time type field. The available time formats are: • "*hms" • "*iso" • "*usa" • "*eur" • "*jis" This property is ignored if the field is not a time field.
timeSeparator	String	The separator for a time type field. The available time separators are: • ':' (colon) • '.' (dot) • ',' (comma) • '&' (ampersand) • ' ' (space) This property is ignored if the field is not a time field and also ignored for time fields with format other than "*hms".
parm	Name	Set to a field name in the calling function's argument list to initialize the field being defined with the argument field value.
refParm	Name	Set to a field name in the calling function's argument list to define the field as a reference to the argument field. The field being defined will be initialized with the argument field value, and any changes made to the field in the current function will affect the field in the calling function.
occurs	Number	This property is used to define a multiple occurrence data structure. Set to the desired number of occurrences. This property is ignored if the field is not a data structure.
from	Number	Used to set the start position for a data structure sub-field. Set to the desired start position within the data structure's buffer. The first position is 1. This property is ignored if the field is not a data structure sub-field.
to	Number	Used to set the end position for a data structure sub-field. Set to the desired end position within the data structure's buffer. This property is ignored if the field is not a data structure sub-field.
ccsid	Number	Sets the CCSID for character, graphic, or UCS-2 fields. This property is ignored for other field types.
rightAdjust	Boolean	For character-based fields only, if this property is true, it will align all the content to the right of the field.
trim	Boolean	For character-based fields only, if this property is true, it will trim the content after the field has been assigned a value. The property has no affect on fields that are not varying length.
static	Boolean	If this property is true, the field will never be redefined. The field will have the same place in memory every time it's used. For example, if you call a function which defines a 'static' variable and then call it again, it won't define a second time because it has already been defined in this scope.
template	Boolean	If this property is true, the field will not actually be defined. Instead, the config and structure of the field will be saved internally so you can reference it with 'like' or 'likeDS' properties later.
nullInd	Boolean / String	If this property is true, the field will have an internal flag which determines whether the field is null or not. The property can also be specified as a string to provide an Indicator field name, which will be used to indicate whether the field is null or not.

prefix	String	This property is only available to the data-structure type. When provided, all the subfields of the data-structure will be prefixed with the property value.
dataArea	String	Name of the data-area object you would like this field to be based over. For example: MYLIB/MYDTAARA
auto	Boolean	This property is only used when a field is based on a data area. When set to true, the data area will automatically be read in when program module starts and automatically closed when the program module ends.
export	String	Allow the variable buffer to be stored in the activation group (by name)
import	String	Receive the buffer from the activation group (by name)
overlay	String	This property can only be used on a subfield of a data-structure. The contents of this subfield will be based on another field name, as specified by this property. A special value of "*next" can be used to get the previous subfield's overlay, add the length of the current subfield, and use that for the overlay.
overlayStart	Number	This property is used together with the overlay property. It provides the ability to specify where the subfield overlay should start, relative to the overlay field position.
noPass	Boolean	This property is used with fields that are passed into the current function. If set to True, the field does not have to be passed. All following parameters must also be defined with 'noPass'.
omit	Boolean	This property is used with fields that are passed into the current function. If set to True, the special value pjs.OMIT can be passed in place of the parameter.
varSize	Boolean	This property is used with fields that are passed into the current function. If set to True, the passed parameter may be shorter or longer in length than is defined. It is then up to the called function to ensure that it accesses only as much data as was passed.
string	Boolean	This property is used with fields that are passed into the current function. If set to True, you may pass either a pointer or a character expression. If you pass a character expression, a temporary value will be created containing the value of the character expression followed by a null-terminator (x'00'). The address of this temporary value will be passed to the called function.
packeven	Boolean	If set to True, the packed field or array will have an even number of digits. This property is only valid for packed data structure subfields defined using from/to positions. For a field or array element of length N, the number of digits will be 2(N-1) when this property is set to True. Otherwise, the number of digits will be 2N - 1.
fractional	Number	This property is used to specify the number of fractional second positions for timestamp fields. Valid values are 0-12. If not specified, the field will be defined with 6 fractional second positions.
table	Boolean	If set to True and the 'dim' option is also used, the field can be initialized by setting the 'initValue' property to an Array.

Examples

Example 1: packed decimal

```
pjs.define("Num", { type: 'packed decimal', length: 3, decimals: 2 });
Num = 5;
```

Example 2: date with initValue

```
pjs.define("loandate", { type: 'date', initValue: pjs.date('2000-01-01') });
```

Example 3: char field with pointer to that field

```
pjs.define("stg", { type: 'char', length: 4 });
pjs.define("ptr", { type: 'pointer', initValue: pjs.getBuffer(stg) });
```

Example 4: char field array with initValue

This does mean that every element in this array will be initilized with the initValue.

```
pjs.define("arr", { type: 'char', length: 10, dim: 3, initValue: '0123456789' });
```

Example 5: Data structure with two subfields (non-qualified)

```
pjs.define("someDS", { type: 'data structure', elements: {
  "Field1": { type: 'char', length: 10, initValue: '1' },
  "Field2": { type: 'char', length: 10 }
}});

Field2 = '2';
```

Example 6: Data structure array, qualified and two subfields

```
pjs.define("dsarr", { type: 'data structure', qualified: true, dim: 2, elements: {
  "field1": { type: 'char', length: 1 },
  "field2": { type: 'char', length: 1 }
}});

dsarr[1].field1 = '1';
dsarr[1].field2 = '2';
dsarr[2].field1 = '3';
dsarr[2].field2 = '4';
```

Example 7: Null-capable field with null assignment

```
pjs.define("myNumber", {
  type: 'packed decimal',
  length: 10,
  decimals: 0,
  nullable: true
});

//Set value to 5
myNumber = 5;

//Set null-indicator to true
myNumber = null;

//Set value to 10, sets the null-indicator to false
myNumber = 10;

//Set null-indicator to true
pjs.nullInd(myNumber, true);
```