:mod:`multiprocessing.shared_memory` --- Provides shared memory for direct access across processes

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 4)

Unknown directive type "module".

```
.. module:: multiprocessing.shared_memory
   :synopsis: Provides shared memory for direct access across processes.
```

Source code: :source:`Lib/multiprocessing/shared_memory.py`

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 7); *backlink*

Unknown interpreted text role "source".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 9)

Unknown directive type "versionadded".

```
.. versionadded:: 3.8
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 11)

Unknown directive type "index".

```
.. index::
   single: Shared Memory
   single: POSIX Shared Memory
   single: Named Shared Memory
```

This module provides a class, :class:`SharedMemory`, for the allocation and management of shared memory to be accessed by one or more processes on a multicore or symmetric multiprocessor (SMP) machine. To assist with the life-cycle management of shared memory especially across distinct processes, a :class:`~multiprocessing.managers.BaseManager` subclass, :class:`SharedMemoryManager`, is also provided in the `multiprocessing.managers` module.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 18); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 18); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst`, line 18); *backlink*

Unknown interpreted text role "class".

In this module, shared memory refers to "System V style" shared memory blocks (though is not necessarily implemented explicitly as such) and does not refer to "distributed shared memory". This style of shared memory permits distinct processes to potentially read and write to a common (or shared) region of volatile memory. Processes are conventionally limited to only have access to their own process memory space but shared memory permits the sharing of data between processes, avoiding the need to instead send messages between processes containing that data. Sharing data directly via memory can provide significant performance benefits compared to sharing data via disk or socket or other communications requiring the serialization/deserialization and copying of data.

Creates a new shared memory block or attaches to an existing shared memory block. Each shared memory block is assigned a unique name. In this way, one process can create a shared memory block with a particular name and a different process can attach to that same shared memory block using that same name.

As a resource for sharing data across processes, shared memory blocks may outlive the original process that created them. When one process no longer needs access to a shared memory block that might still be needed by other processes, the :meth:`close()` method should be called. When a shared memory block is no longer needed by any process, the :meth:`unlink()` method should be called to ensure proper cleanup.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 47); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 47); *backlink*

Unknown interpreted text role "meth".

name is the unique name for the requested shared memory, specified as a string. When creating a new shared memory block, if `None` (the default) is supplied for the name, a novel name will be generated.

create controls whether a new shared memory block is created (`True`) or an existing shared memory block is attached (`False`).

size specifies the requested number of bytes when creating a new shared memory block. Because some platforms choose to allocate chunks of memory based upon that platform's memory page size, the exact size of the shared memory block may be larger or equal to the size requested. When attaching to an existing shared memory block, the `size` parameter is ignored.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 67)

Unknown directive type "method".

```
.. method:: close()

   Closes access to the shared memory from this instance.  In order to
   ensure proper cleanup of resources, all instances should call
   ``close()`` once the instance is no longer needed.  Note that calling
   ``close()`` does not cause the shared memory block itself to be
   destroyed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 75)

Unknown directive type "method".

```
.. method:: unlink()

   Requests that the underlying shared memory block be destroyed.  In
   order to ensure proper cleanup of resources, ``unlink()`` should be
   called once (and only once) across all processes which have need
   for the shared memory block.  After requesting its destruction, a
   shared memory block may or may not be immediately destroyed and
   this behavior may differ across platforms.  Attempts to access data
   inside the shared memory block after ``unlink()`` has been called may
   result in memory access errors.  Note: the last process relinquishing
   its hold on a shared memory block may call ``unlink()`` and
   :meth:`close()` in either order.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 88)

Unknown directive type "attribute".

```
.. attribute:: buf

   A memoryview of contents of the shared memory block.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 92)

Unknown directive type "attribute".

```
.. attribute:: name

   Read-only access to the unique name of the shared memory block.
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst, line 96)

Unknown directive type "attribute".

```
.. attribute:: size

   Read-only access to size in bytes of the shared memory block.
```

The following example demonstrates low-level use of :class:`SharedMemory` instances:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst, line 101); *backlink*

Unknown interpreted text role "class".

```
>>> from multiprocessing import shared_memory
>>> shm_a = shared_memory.SharedMemory(create=True, size=10)
>>> type(shm_a.buf)
<class 'memoryview'>
>>> buffer = shm_a.buf
>>> len(buffer)
10
>>> buffer[:4] = bytearray([22, 33, 44, 55])  # Modify multiple at once
>>> buffer[4] = 100                           # Modify single byte at a time
>>> # Attach to an existing shared memory block
>>> shm_b = shared_memory.SharedMemory(shm_a.name)
>>> import array
>>> array.array('b', shm_b.buf[:5])  # Copy the data into a new array.array
array('b', [22, 33, 44, 55, 100])
>>> shm_b.buf[:5] = b'howdy'  # Modify via shm_b using bytes
>>> bytes(shm_a.buf[:5])      # Access via shm_a
b'howdy'
>>> shm_b.close()   # Close each SharedMemory instance
>>> shm_a.close()
>>> shm_a.unlink()  # Call unlink only once to release the shared memory
```

The following example demonstrates a practical use of the :class:`SharedMemory` class with NumPy arrays, accessing the same `numpy.ndarray` from two distinct Python shells:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst, line 127); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.shared_memory.rst, line 131)

Unknown directive type "doctest".

```
.. doctest::
   :options: +SKIP

   >>> # In the first Python interactive shell
   >>> import numpy as np
   >>> a = np.array([1, 1, 2, 3, 5, 8])  # Start with an existing NumPy array
   >>> from multiprocessing import shared_memory
   >>> shm = shared_memory.SharedMemory(create=True, size=a.nbytes)
   >>> # Now create a NumPy array backed by shared memory
   >>> b = np.ndarray(a.shape, dtype=a.dtype, buffer=shm.buf)
   >>> b[:] = a[:]  # Copy the original data into shared memory
   >>> b
   array([1, 1, 2, 3, 5, 8])
   >>> type(b)
   <class 'numpy.ndarray'>
   >>> type(a)
   <class 'numpy.ndarray'>
   >>> shm.name  # We did not specify a name so one was chosen for us
   'psm_21467_46075'

   >>> # In either the same shell or a new Python shell on the same machine
   >>> import numpy as np
   >>> from multiprocessing import shared_memory
   >>> # Attach to the existing shared memory block
   >>> existing_shm = shared_memory.SharedMemory(name='psm_21467_46075')
   >>> # Note that a.shape is (6,) and a.dtype is np.int64 in this example
   >>> c = np.ndarray((6,), dtype=np.int64, buffer=existing_shm.buf)
   >>> c
   array([1, 1, 2, 3, 5, 8])
   >>> c[-1] = 888
   >>> c
   array([  1,   1,   2,   3,   5, 888])

   >>> # Back in the first Python interactive shell, b reflects this change
   >>> b
   array([  1,   1,   2,   3,   5, 888])
```

```
>>> # Clean up from within the second Python shell
>>> del c  # Unnecessary; merely emphasizing the array is no longer used
>>> existing_shm.close()

>>> # Clean up from within the first Python shell
>>> del b  # Unnecessary; merely emphasizing the array is no longer used
>>> shm.close()
>>> shm.unlink()  # Free and release the shared memory block at the very end
```

A subclass of :class:`~multiprocessing.managers.BaseManager` which can be used for the management of shared memory blocks across processes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 181); *backlink*

Unknown interpreted text role "class".

A call to :meth:`~multiprocessing.managers.BaseManager.start` on a :class:`SharedMemoryManager` instance causes a new process to be started. This new process's sole purpose is to manage the life cycle of all shared memory blocks created through it. To trigger the release of all shared memory blocks managed by that process, call :meth:`~multiprocessing.managers.BaseManager.shutdown()` on the instance. This triggers a :meth:`SharedMemory.unlink()` call on all of the :class:`SharedMemory` objects managed by that process and then stops the process itself. By creating `SharedMemory` instances through a `SharedMemoryManager`, we avoid the need to manually track and trigger the freeing of shared memory resources.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 184); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 184); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 184); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 184); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 184); *backlink*

Unknown interpreted text role "class".

This class provides methods for creating and returning :class:`SharedMemory` instances and for creating a list-like object (:class:`ShareableList`) backed by shared memory.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 196); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 196); *backlink*

Unknown interpreted text role "class".

Refer to :class:`multiprocessing.managers.BaseManager` for a description of the inherited *address* and *authkey* optional input arguments and how they may be used to connect to an existing `SharedMemoryManager` service from other processes.

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 200); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 205)

Unknown directive type "method".

```
.. method:: SharedMemory(size)

   Create and return a new :class:`SharedMemory` object with the
   specified ``size`` in bytes.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 210)

Unknown directive type "method".

```
.. method:: ShareableList(sequence)

   Create and return a new :class:`ShareableList` object, initialized
   by the values from the input ``sequence``.
```

The following example demonstrates the basic mechanisms of a :class:`SharedMemoryManager`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 216); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 219)

Unknown directive type "doctest".

```
.. doctest::
   :options: +SKIP

   >>> from multiprocessing.managers import SharedMemoryManager
   >>> smm = SharedMemoryManager()
   >>> smm.start()  # Start the process that manages the shared memory blocks
   >>> sl = smm.ShareableList(range(4))
   >>> sl
   ShareableList([0, 1, 2, 3], name='psm_6572_7512')
   >>> raw_shm = smm.SharedMemory(size=128)
   >>> another_sl = smm.ShareableList('alpha')
   >>> another_sl
   ShareableList(['a', 'l', 'p', 'h', 'a'], name='psm_6572_12221')
   >>> smm.shutdown()  # Calls unlink() on sl, raw_shm, and another_sl
```

The following example depicts a potentially more convenient pattern for using :class:`SharedMemoryManager` objects via the :keyword:`with` statement to ensure that all shared memory blocks are released after they are no longer needed:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 234); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 234); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 239)

Unknown directive type "doctest".

```
.. doctest::
   :options: +SKIP

   >>> with SharedMemoryManager() as smm:
   ...     sl = smm.ShareableList(range(2000))
   ...     # Divide the work among two processes, storing partial results in sl
   ...     p1 = Process(target=do_work, args=(sl, 0, 1000))
```

```
...     p2 = Process(target=do_work, args=(sl, 1000, 2000))
...     p1.start()
...     p2.start()  # A multiprocessing.Pool might be more efficient
...     p1.join()
...     p2.join()   # Wait for all work to complete in both processes
...     total_result = sum(sl)  # Consolidate the partial results now in sl
```

When using a :class:`SharedMemoryManager` in a :keyword:`with` statement, the shared memory blocks created using that manager are all released when the :keyword:`with` statement's code block finishes execution.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 253); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 253); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 253); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 258)

Invalid class attribute value for "class" directive: "ShareableList(sequence=None, *, name=None)".

```
.. class:: ShareableList(sequence=None, *, name=None)

   Provides a mutable list-like object where all values stored within are
   stored in a shared memory block.  This constrains storable values to
   only the ``int``, ``float``, ``bool``, ``str`` (less than 10M bytes each),
   ``bytes`` (less than 10M bytes each), and ``None`` built-in data types.
   It also notably differs from the built-in ``list`` type in that these
   lists can not change their overall length (i.e. no append, insert, etc.)
   and do not support the dynamic creation of new :class:`ShareableList`
   instances via slicing.

   *sequence* is used in populating a new ``ShareableList`` full of values.
   Set to ``None`` to instead attach to an already existing
   ``ShareableList`` by its unique shared memory name.

   *name* is the unique name for the requested shared memory, as described
   in the definition for :class:`SharedMemory`.  When attaching to an
   existing ``ShareableList``, specify its shared memory block's unique
   name while leaving ``sequence`` set to ``None``.

   .. method:: count(value)

      Returns the number of occurrences of ``value``.

   .. method:: index(value)

      Returns first index position of ``value``.  Raises :exc:`ValueError` if
      ``value`` is not present.

   .. attribute:: format

      Read-only attribute containing the :mod:`struct` packing format used by
      all currently stored values.

   .. attribute:: shm

      The :class:`SharedMemory` instance where the values are stored.
```

The following example demonstrates basic use of a :class:`ShareableList` instance:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 297); *backlink*

Unknown interpreted text role "class".

```
>>> from multiprocessing import shared_memory
>>> a = shared_memory.ShareableList(['howdy', b'HoWdY', -273.154, 100, None, True, 42])
>>> [ type(entry) for entry in a ]
```

```
[<class 'str'>, <class 'bytes'>, <class 'float'>, <class 'int'>, <class 'NoneType'>, <class 'bool'>, <class
>>> a[2]
-273.154
>>> a[2] = -78.5
>>> a[2]
-78.5
>>> a[2] = 'dry ice'  # Changing data types is supported as well
>>> a[2]
'dry ice'
>>> a[2] = 'larger than previously allocated storage space'
Traceback (most recent call last):
  ...
ValueError: exceeds available storage for existing str
>>> a[2]
'dry ice'
>>> len(a)
7
>>> a.index(42)
6
>>> a.count(b'howdy')
0
>>> a.count(b'HoWdY')
1
>>> a.shm.close()
>>> a.shm.unlink()
>>> del a  # Use of a ShareableList after call to unlink() is unsupported
```

The following example depicts how one, two, or many processes may access the same :class:`ShareableList` by supplying the name of the shared memory block behind it:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.shared_memory.rst, line 330); *backlink*

Unknown interpreted text role "class".

```
>>> b = shared_memory.ShareableList(range(5))         # In a first process
>>> c = shared_memory.ShareableList(name=b.shm.name)  # In a second process
>>> c
ShareableList([0, 1, 2, 3, 4], name='...')
>>> c[-1] = -999
>>> b[-1]
-999
>>> b.shm.close()
>>> c.shm.close()
>>> c.shm.unlink()
```

The following examples demonstrates that `ShareableList` (and underlying `SharedMemory`) objects can be pickled and unpickled if needed. Note, that it will still be the same shared object. This happens, because the deserialized object has the same unique name and is just attached to an existing object with the same name (if the object is still alive):

```
>>> import pickle
>>> from multiprocessing import shared_memory
>>> sl = shared_memory.ShareableList(range(10))
>>> list(sl)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> deserialized_sl = pickle.loads(pickle.dumps(sl))
>>> list(deserialized_sl)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> sl[0] = -1
>>> deserialized_sl[1] = -2
>>> list(sl)
[-1, -2, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(deserialized_sl)
[-1, -2, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> sl.shm.close()
>>> sl.shm.unlink()
```