

IRfile:compat.py

Link

[tensorflow/tensorflow/python/util/compat.py at v2.13.0 · tensorflow/tensorflow · GitHub](#)

▼ Code

```
import numbers as _numbers

import numpy as _np
import six as _six
import codecs

from tensorflow.python.util.tf_export import tf_export

try:
    # This import only works on python 3.3 and above.
    import collections.abc as collections_abc # pylint: disable=import-error
except ImportError:
    import collections as collections_abc # pylint: disable=import-error

def as_bytes(bytes_or_text, encoding='utf-8'):
    """Converts `bytearray`, `bytes`, or unicode python input to bytes.

    Uses utf-8 encoding for text by default.

    Args:
        bytes_or_text: A `bytearray`, `bytes`, `str`, or `unicode` object.
        encoding: A string indicating the charset for encoding un
```

Returns:

A ``bytes`` object.

Raises:

`TypeError`: If ``bytes_or_text`` is not a binary or unicode
"""

Validate encoding, a `LookupError` will be raised if invalid

`encoding = codecs.lookup(encoding).name`

`if isinstance(bytes_or_text, bytearray):`

`return bytes(bytes_or_text)`

`elif isinstance(bytes_or_text, _six.text_type):`

`return bytes_or_text.encode(encoding)`

`elif isinstance(bytes_or_text, bytes):`

`return bytes_or_text`

`else:`

`raise TypeError('Expected binary or unicode string, got %`
`(bytes_or_text,))`

`def as_text(bytes_or_text, encoding='utf-8'):`

"""Converts any string-like python input types to unicode.

Returns the input as a unicode string. Uses utf-8 encoding
by default.

Args:

`bytes_or_text`: A ``bytes``, ``str``, or ``unicode`` object.

`encoding`: A string indicating the charset for decoding un

Returns:

A ``unicode`` (Python 2) or ``str`` (Python 3) object.

Raises:

`TypeError`: If ``bytes_or_text`` is not a binary or unicode
"""

Validate encoding, a `LookupError` will be raised if invalid

```

encoding = codecs.lookup(encoding).name
if isinstance(bytes_or_text, _six.text_type):
    return bytes_or_text
elif isinstance(bytes_or_text, bytes):
    return bytes_or_text.decode(encoding)
else:
    raise TypeError('Expected binary or unicode string, got %

def as_str(bytes_or_text, encoding='utf-8'):
    return as_text(bytes_or_text, encoding)

tf_export('compat.as_text')(as_text)
tf_export('compat.as_bytes')(as_bytes)
tf_export('compat.as_str')(as_str)

@tf_export('compat.as_str_any')
def as_str_any(value, encoding='utf-8'):
    """Converts input to `str` type.

    Uses `str(value)`, except for `bytes` typed inputs, which
    using `as_str`.

    Args:
        value: A object that can be converted to `str`.
        encoding: Encoding for `bytes` typed inputs.

    Returns:
        A `str` object.
    """
    if isinstance(value, bytes):
        return as_str(value, encoding=encoding)
    else:
        return str(value)

```

```

@tf_export('compat.path_to_str')
def path_to_str(path):
    r"""Converts input which is a `PathLike` object to `str` type.

    Converts from any python constant representation of a `Path`
    to a string. If the input is not a `PathLike` object, simply return
    the input.

    Args:
        path: An object that can be converted to path representation.

    Returns:
        A `str` object.

    Usage:
        In case a simplified `str` version of the path is needed, use
        `os.PathLike` object.

    Examples:
    ```python
 $ tf.compat.path_to_str('C:\XYZ\tensorflow\../../tensorflow')
 'C:\XYZ\tensorflow\../../tensorflow' # Windows OS
 $ tf.compat.path_to_str(Path('C:\XYZ\tensorflow\../../tensorflow'))
 'C:\XYZ\tensorflow\..\tensorflow' # Windows OS
 $ tf.compat.path_to_str(Path('../corpus'))
 'corpus' # Linux OS
 $ tf.compat.path_to_str('../../Corpus')
 '../../Corpus' # Linux OS
 $ tf.compat.path_to_str(Path('../../Corpus'))
 '../Corpus' # Linux OS
 $ tf.compat.path_to_str(Path('../..//..//..'))
 '../..' # Linux OS
 """
 if hasattr(path, '__fspath__'):

```

```

 path = as_str_any(path.__fspath__())
 return path

def path_to_bytes(path):
 r"""Converts input which is a `PathLike` object to `bytes`.

 Converts from any python constant representation of a `Path`
 or `str` to bytes.

 Args:
 path: An object that can be converted to path representat

 Returns:
 A `bytes` object.

 Usage:
 In case a simplified `bytes` version of the path is needed
 `os.PathLike` object.
 """
 if hasattr(path, '__fspath__'):
 path = path.__fspath__()
 return as_bytes(path)

Numpy 1.8 scalars don't inherit from numbers.Integral in Py
need to check them specifically. The same goes from Real a
integral_types = (_numbers.Integral, _np.integer)
tf_export('compat.integral_types').export_constant(__name__,
real_types = (_numbers.Real, _np.integer, _np.floating)
tf_export('compat.real_types').export_constant(__name__, 'rea
complex_types = (_numbers.Complex, _np.number)
tf_export('compat.complex_types').export_constant(__name__, '

Either bytes or text.
bytes_or_text_types = (bytes, _six.text_type)

```

```
tf_export('compat.bytes_or_text_types').export_constant(__name__,
 'byte
```

## ▼ Documentation

```
Copyright 2015 The TensorFlow Authors. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License";
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing,
distributed under the License is distributed on an "AS IS"
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either expressed or
implied. See the License for the specific language governing permissions
and limitations under the License.
=====
"""Compatibility functions.
```

The ``tf.compat`` module contains two sets of compatibility functions.

```
Tensorflow 1.x and 2.x APIs
```

The ``compat.v1`` and ``compat.v2`` submodules provide a complete set of ``v1`` and ``v2`` APIs for backwards and forwards compatibility across TensorFlow versions 1.x and 2.x. See the [\[migration guide\]\(https://www.tensorflow.org/guide/migrate\)](https://www.tensorflow.org/guide/migrate) for more details.

```
Utilities for writing compatible code
```

Aside from the ``compat.v1`` and ``compat.v2`` submodules, ``tf.compat`` also contains a set of helper functions for writing code that works in both TensorFlow 1.x and 2.x.

```
* TensorFlow 1.x and 2.x
* Python 2 and 3
```

```
Type collections
```

The compatibility module also provides the following aliases sets of python types:

```
* `bytes_or_text_types`
* `complex_types`
* `integral_types`
* `real_types`
"""
```

---

## Prompt

As a code documentation assistant, you are responsible for documenting at the file/script level. When provided a file level code, your approach involves generating a header comment to be added at the top of the file. This comment should be the documentation for the code and include all relevant information needed to understand or use the script. The code is provided below.

---

## Responses

ChatGPT

```
"""
```

## Compatibility Utilities

This module provides utility functions for handling compatibility between different Python versions and data types. It includes functions for converting between byte arrays, text, and strings, as well as handling conversion and defining compatibility types.

### Functions:

- ``as_bytes(bytes_or_text, encoding='utf-8')``: Converts various to bytes.
- ``as_text(bytes_or_text, encoding='utf-8')``: Converts various : text (unicode).
- ``as_str(bytes_or_text, encoding='utf-8')``: Alias for ``as_text``
- ``as_str_any(value, encoding='utf-8')``: Converts input to a str
- ``path_to_str(path)``: Converts a ``PathLike`` object to a string
- ``path_to_bytes(path)``: Converts a ``PathLike`` object or string
- Constants for integral, real, and complex types.
- Constants for bytes or text types.

### Compatibility:

- Python 2 and 3 compatibility is ensured through the use of the
- Numpy compatibility is handled for different scalar types.

### Usage:

This module is intended to be used when dealing with compatibility between Python versions or when working with different data types. It provides consistent conversion functions and type constants to simplify development.

```
"""
```

## GPT4



```
"""
```

This script provides utility functions for type conversion and compatibility within TensorFlow, ensuring support across different Python versions and compatibility with numpy and TensorFlow-specific types.

It includes functions to convert between bytes, text, and strings, and is compatible with both Python 2 and 3. Additionally, the script defines type groups like `integral`, `real`, `complex`, and `bytes_or_text_types` for TensorFlow's compatibility layer.

#### Key Functions:

- `as_bytes(bytes_or_text, encoding='utf-8')`: Converts bytearray, bytes, or text to bytes.
- `as_text(bytes_or_text, encoding='utf-8')`: Converts bytes or unicode (Python 2) or str (Python 3) to text.
- `as_str(bytes_or_text, encoding='utf-8')`: Convenience function to convert bytes or text to a string.
- `as_str_any(value, encoding='utf-8')`: Converts any input to a string, handling bytes and text.
- `path_to_str(path)`: Converts a PathLike object to a str.
- `path_to_bytes(path)`: Converts a PathLike object or str to bytes.

#### Type Groups:

- `integral_types`: Group combining Python's Integral and numpy's integer types.
- `real_types`: Group combining Python's Real, numpy's integer and float types.
- `complex_types`: Group combining Python's Complex and numpy's complex types.
- `bytes_or_text_types`: Group for bytes and text types.

#### Dependencies:

- Python 3.3+ (for certain imports)
- TensorFlow
- NumPy
- Six (Python 2 and 3 compatibility library)

#### Note:

- The script is intended to be used within the TensorFlow framework.

```
exports using tf_export for TensorFlow's compatibility module
```

```
Author: [Your Name or Team Name]
```

```
Date: [Creation Date]
```

```
"""
```

## Bard

```
"""
```

This module provides utilities for converting between Python's primitive types (strings, numbers, and booleans) and TensorFlow Tensor representations.

The main conversions provided here are:

- \* ``as_bytes``: Converts a ``str`` or ``bytes`` to a ``bytes``.
- \* ``as_text``: Converts a ``bytes`` or ``str`` to a ``str``.
- \* ``as_str_any``: Converts any Python object to a ``str``.
- \* ``path_to_str``: Converts a ``PathLike`` object to a ``str``.
- \* ``path_to_bytes``: Converts a ``PathLike`` object to a ``bytes``.

In addition, this module provides the following constants:

- \* ``integral_types``: A list of all integral types.
  - \* ``real_types``: A list of all real types (integral and floating
  - \* ``complex_types``: A list of all complex types.
  - \* ``bytes_or_text_types``: A list of all bytes or text types.
- ```
"""
```

LLama2

This code defines various utility functions for working with di

1. ``as_str``: This function converts its argument to a string. If
2. ``as_str_any``: This function converts its argument to a string

3. ``path_to_str``: This function converts a PathLike object to a string
4. ``path_to_bytes``: This function converts a PathLike object or string to bytes
5. ``integral_types``: This variable exports a tuple of integral types
6. ``real_types``: This variable exports a tuple of real types, including float
7. ``complex_types``: This variable exports a tuple of complex types
8. ``bytes_or_text_types``: This variable exports a tuple of types that can be used for string-like operations

These utilities provide convenient ways to work with different types of paths and types.

StarChat

```
/**
 * Copyright [YEAR] Google LLC
 * Licensed under the Apache License, Version 2.0 (the "License";
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```