```
BIP: 381
Layer: Applications
Title: Non-Segwit Output Script Descriptors
Author: Pieter Wuille <pieter@wuille.net>
        Andrew Chow <andrew@achow101.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0381
Status: Draft
Type: Informational
Created: 2021-06-27
License: BSD-2-Clause
```

Abstract

This document specifies `pk()`, `pkh()`, and `sh()` output script descriptors. `pk()` descriptors take a key and produces a P2PK output script. `pkh()` descriptors take a key and produces a P2PKH output script. `sh()` descriptors take a script and produces a P2SH output script.

Copyright

This BIP is licensed under the BSD 2-clause license.

Motivation

Prior to the activation of Segregated Witness, there were 3 main standard output script formats: P2PK, P2PKH, and P2SH. These expressions allow specifying those formats as a descriptor.

Specification

Three new script expressions are defined: `pk()`, `pkh()`, and `sh()`.

`pk()`

The `pk(KEY)` expression can be used in any context or level of a descriptor. It takes a single key expression as an argument and produces a P2PK output script. Depending on the higher level descriptors, there may be restrictions on the type of public keys that can be included. Such restrictions will be specified by those descriptors.

The output script produced is:

```
<KEY> OP_CHECKSIG
```

pkh()

The pkh(KEY) expression can be used as a top level expression, or inside of either a sh() or wsh() descriptor. It takes a single key expression as an argument and produces a P2PKH output script. Depending on the higher level descriptors, there may be restrictions on the type of public keys that can be included. Such restrictions will be specified by those descriptors.

The output script produced is:

```
OP_DUP OP_HASH160 <KEY_hash160> OP_EQUALVERIFY OP_CHECKSIG
```

sh()

The sh(SCRIPT) expression can only be used as a top level expression. It takes a single script expression as an argument and produces a P2SH output script. sh() expressions also create a redeemScript which is required in order to spend outputs which use its output script. This redeemScript is the output script produced by the SCRIPT argument to sh().

The output script produced is:

```
OP_HASH160 <SCRIPT_hash160> OP_EQUAL
```

Test Vectors

Valid descriptors followed by the scripts they produce. Descriptors involving derived child keys will have the 0th, 1st, and 2nd scripts listed.

- pk(L4rK1yDtCWekvXuE6oXD9jCYfFNV2cWRpVuPLBcCU2z8TrisoyY1)
  - 2103a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bdac
- pk(03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bd)
  - 2103a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bdac
- pkh([deadbeef/1/2'/3/4']L4rK1yDtCWekvXuE6oXD9jCYfFNV2cWRpVuPLBcCU2z8TrisoyY1)
  - 76a9149a1c78a507689f6f54b847ad1cef1e614ee23f1e88ac
- pkh([deadbeef/1/2'/3/4']03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c
  - 76a9149a1c78a507689f6f54b847ad1cef1e614ee23f1e88ac
- pkh([deadbeef/1/2h/3/4h]03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c
  - 76a9149a1c78a507689f6f54b847ad1cef1e614ee23f1e88ac
- pk(5KYZdUEo39z3FPrtuX2QbbwGnNP5zTd7yyr2SC1j299sBCnWjss)
  - 4104a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bd5b8dec5235a0fa8
- pk(04a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bd5b8dec5235a0fa8722
  - 4104a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bd5b8dec5235a0fa8
- pkh(5KYZdUEo39z3FPrtuX2QbbwGnNP5zTd7yyr2SC1j299sBCnWjss)
  - 76a914b5bd079c4d57cc7fc28ecf8213a6b791625b818388ac
- pkh(04a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bd5b8dec5235a0fa872
  - 76a914b5bd079c4d57cc7fc28ecf8213a6b791625b818388ac
- sh(pk(L4rK1yDtCWekvXuE6oXD9jCYfFNV2cWRpVuPLBcCU2z8TrisoyY1))
  - a9141857af51a5e516552b3086430fd8ce55f7c1a52487

- `sh(pk(03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bd))`
  - `a9141857af51a5e516552b3086430fd8ce55f7c1a52487`
- `sh(pkh(L4rK1yDtCWekvXuE6oXD9jCYfFNV2cWRpVuPLBcCU2z8TrisoyY1))`
  - `a9141a31ad23bf49c247dd531a623c2ef57da3c400c587`
- `sh(pkh(03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c540c5bd))`
  - `a9141a31ad23bf49c247dd531a623c2ef57da3c400c587`
- `pkh(xprv9s21ZrQH143K31xYSDQpPDxsXRTUcvj2iNHm5NUtrGiGG5e2DtALGdso3pGz6ssrdK4PFmM8NSpSBHN`
  - `76a914ebdc90806a9c4356c1c88e42216611e1cb4c1c1788ac`
- `pkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCGu8ED9W6oDMSgv6Zz8idoc4a`
  - `76a914ebdc90806a9c4356c1c88e42216611e1cb4c1c1788ac`
- `pkh([bd16bee5/2147483647h]xpub69H7F5dQzmVd3vPuLKtcXJziMEQByuDidnX3YdwgtNsecY5HRGtAAQC5m`
  - `76a914ebdc90806a9c4356c1c88e42216611e1cb4c1c1788ac`
- `pk(xprv9uPDJpEQgRQfDcW7BkF7eTya6RPxXeJCqCJGHuCJ4GiRVLzkTXBAJMu2qaMWPrS7AANYqdq6vcBcBUdJ`
  - `210379e45b3cf75f9c5f9befd8e9506fb962f6a9d185ac87001ec44a8d3df8d4a9e3ac`
- `pk(xpub68NZiKmJWnxxS6aaHmn81bvJeTESw724CRDs6HbuccFQN9Ku14VQrADWgqbhhTHBaohPX4CjNLf9fq9M`
  - `210379e45b3cf75f9c5f9befd8e9506fb962f6a9d185ac87001ec44a8d3df8d4a9e3ac`

Invalid descriptors

- `pk()` only accepts key expressions: `pk(pk(03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b`
- `pkh()` only accepts key expressions: `pkh(pk(03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b`
- `sh()` only acceps script expressions: `sh(03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56`
- `sh()` is top level only: `sh(sh(pkh(03a34b99f22c790c4e36b2b3c2c35a36db06226e41c692fc82b8b56ac1c`

Backwards Compatibility

`pk()`, `pkh()`, and `sh()` descriptors use the format and general operation specified in 380. As these are a wholly new descriptors, they are not compatible with any implementation. However the scripts produced are standard scripts so existing software are likely to be familiar with them.

Reference Implementation

`pk()`, `pkh()`, and `sh()` descriptors have been implemented in Bitcoin Core since version 0.17.