# OnchainKit Complete Reference Guide

## Core Setup

### Basic Installation
```bash
npm install @coinbase/onchainkit
```

### Required Configuration
Core provider setup needed for all OnchainKit functionality:

```typescript
import { OnchainKitProvider } from '@coinbase/onchainkit';
import { base } from 'wagmi/chains';

<OnchainKitProvider
  apiKey={process.env.NEXT_PUBLIC_ONCHAINKIT_API_KEY}
  chain={base}
  config={{
    appearance: {
      name: "Your App Name",
      logo: "https://your-logo-url.com/logo.png",
      mode: "auto", // "auto" | "light" | "dark"
      theme: "default" // "default" | "base" | "cyberpunk" | "hacker"
    }
  }}
>
  {children}
</OnchainKitProvider>
```

## Components

### Identity Components

#### `<Avatar />`
Displays ENS or Basename avatar associated with Ethereum addresses.

```typescript
import { Avatar } from '@coinbase/onchainkit/identity';

// Basic usage
<Avatar address="0x123..." />

// With chain specification
<Avatar address="0x123..." chain={base} />

// With custom styling
<Avatar
  address="0x123..."
  className="h-6 w-6"
  loadingComponent={<LoadingSpinner />}
  defaultComponent={<DefaultAvatar />}
/>
```

#### `<Name />`
Displays ENS or Basename associated with Ethereum addresses.

```typescript
import { Name } from '@coinbase/onchainkit/identity';

// Basic usage
<Name address="0x123..." />

// With chain specification
<Name address="0x123..." chain={base} />

// With custom styling
<Name
  address="0x123..."
  className="text-lg font-bold"
/>
```

#### `<Address />`
Displays formatted Ethereum addresses.

```typescript
import { Address } from '@coinbase/onchainkit/identity';

// Basic usage
<Address address="0x123..." />

// With slicing disabled
<Address address="0x123..." isSliced={false} />

// With custom styling
<Address
  address="0x123..."
  className="text-gray-600"
/>
```

### Wallet Components

#### `<Wallet />`
Main container for wallet functionality.

```typescript
import {
  Wallet,
  ConnectWallet,
  WalletDropdown,
  WalletDropdownDisconnect
} from '@coinbase/onchainkit/wallet';

<Wallet>
  <ConnectWallet>
    <Avatar className="h-6 w-6" />
    <Name />
  </ConnectWallet>
  <WalletDropdown>
    <Identity className="px-4 pt-3 pb-2" hasCopyAddressOnClick>
```

~~~
      <Avatar />
      <Name />
      <Address />
      <EthBalance />
    </Identity>
    <WalletDropdownDisconnect />
  </WalletDropdown>
</Wallet>
```

### Transaction Components

#### `<Transaction />`
Handles complete transaction lifecycle.

```typescript
import {
  Transaction,
  TransactionButton,
  TransactionStatus
} from '@coinbase/onchainkit/transaction';

<Transaction
  contracts={[{
    address: contractAddress,
    abi: contractAbi,
    functionName: 'functionToCall',
    args: [arg1, arg2]
  }]}
  onStatus={(status) => console.log(status)}
>
  <TransactionButton />
  <TransactionStatus />
</Transaction>
```

### Swap Components

#### `<Swap />`
Handles token swap functionality.

```typescript
import {
  Swap,
  SwapAmountInput,
  SwapButton,
  SwapMessage
} from '@coinbase/onchainkit/swap';

<Swap>
  <SwapAmountInput
    label="Sell"
    swappableTokens={tokens}
    token={fromToken}
    type="from"
  />
  <SwapToggleButton />
  <SwapAmountInput
~~~

```
    label="Buy"
    swappableTokens={tokens}
    token={toToken}
    type="to"
  />
  <SwapButton />
  <SwapMessage />
</Swap>
```

## Utilities

### Identity Utilities

#### `getName`
Retrieves name from onchain identity provider.

```typescript
import { getName } from '@coinbase/onchainkit/identity';

const name = await getName({
  address: "0x123...",
  chain: base // optional
});
```

#### `getAddress`
Retrieves address from onchain identity provider.

```typescript
import { getAddress } from '@coinbase/onchainkit/identity';

const address = await getAddress({
  name: "name.eth",
  chain: base // optional
});
```

### Frame Utilities

#### `getFrameMetadata`
Generates Frame metadata for use in Next.js.

```typescript
import { getFrameMetadata } from '@coinbase/onchainkit/frame';

const frameMetadata = getFrameMetadata({
  buttons: [
    { label: "Button 1" },
    { label: "Button 2", action: "link", target: "https://example.com" }
  ],
  image: "https://example.com/image.png",
  postUrl: "https://api.example.com/frame"
});
```

#### `getFrameMessage`
Validates and processes Frame interaction messages.

```typescript
import { getFrameMessage } from '@coinbase/onchainkit/frame';

const { isValid, message } = await getFrameMessage(frameRequest, {
  neynarApiKey: "optional-key",
  castReactionContext: true,
  followContext: true
});
```

### API Utilities

#### `getTokens`
Retrieves list of tokens on Base.

```typescript
import { getTokens } from '@coinbase/onchainkit/api';

const tokens = await getTokens({
  limit: "50",
  page: "1",
  search: "USDC"
});
```

#### `getSwapQuote`
Gets quote for token swap.

```typescript
import { getSwapQuote } from '@coinbase/onchainkit/api';

const quote = await getSwapQuote({
  from: fromToken,
  to: toToken,
  amount: "0.1",
  useAggregator: false
});
```

## Type Definitions

### Common Types

```typescript
type Token = {
  address: string;
  chainId: number;
  decimals: number;
  image: string | null;
  name: string;
  symbol: string;
};

type LifecycleStatus = {
  statusName: 'init' | 'error' | 'success' | /* other status names */;
  statusData: any; // Varies by status
};
```

type Transaction = {
  chainId: number;
  data: string;
  gas: bigint;
  to: string;
  value: bigint;
};

## Error Handling

All components accept `onError` callbacks for handling errors:

```typescript
<Component
  onError={(error) => {
    console.error(error.code, error.message);
    // Handle error appropriately
  }}
/>
```

## Best Practices

1. Always wrap your app with `OnchainKitProvider`
2. Handle all lifecycle status changes through `onStatus` callbacks
3. Use appropriate error handling for all async operations
4. Follow proper component hierarchy (e.g., `Wallet` containing `ConnectWallet`)
5. Implement proper type checking using provided TypeScript definitions

## Common Patterns

### Wallet Connection Flow
```typescript
<Wallet>
  <ConnectWallet>
    <Avatar className="h-6 w-6" />
    <Name />
  </ConnectWallet>
  {isConnected && (
    <WalletDropdown>
      <Identity hasCopyAddressOnClick>
        <Avatar />
        <Name />
        <Address />
      </Identity>
      <WalletDropdownDisconnect />
    </WalletDropdown>
  )}
</Wallet>
```

### Transaction Flow
```typescript
<Transaction
  contracts={contracts}
  onStatus={(status) => {
    switch(status.statusName) {
      case 'success':
        // Handle success
        break;
      case 'error':
        // Handle error
        break;
      // Handle other states
    }
  }}
>
  <TransactionButton />
  <TransactionStatus />
</Transaction>
```

# OnchainKit Components Complete Reference

## Identity Components

### `<Avatar />`
Displays ENS or Basename avatar associated with Ethereum addresses.

Props (`AvatarReact`):
```typescript
type AvatarReact = {
  address?: Address | null;         // The Ethereum address to fetch the avatar for
  chain?: Chain;                    // Optional chain for domain resolution
  className?: string;               // Optional className override for top div element
  loadingComponent?: JSX.Element;   // Optional custom loading component
  defaultComponent?: JSX.Element;   // Optional custom default component when no avatar
  children?: ReactNode;             // Optional attestation badge
};
```

Example usage:
```typescript
// Basic usage
<Avatar address="0x123..." />

// With loading and default states
<Avatar
  address="0x123..."
  chain={base}
  className="h-12 w-12 rounded-full"
  loadingComponent={<Spinner />}
  defaultComponent={<DefaultAvatar />}
>
  <Badge /> // Optional attestation badge
</Avatar>
```

### `<Name />`
Displays ENS or Basename associated with addresses.

Props (`NameReact`):

```typescript
type NameReact = {
  address?: Address | null;     // Ethereum address to be displayed
  children?: ReactNode;         // Optional attestation badge
  chain?: Chain;               // Optional chain for domain resolution
  className?: string;          // Optional className override
} & HTMLAttributes<HTMLSpanElement>;
```

Example usage:
```typescript
// Basic usage
<Name address="0x123..." />

// With attestation and custom styling
<Name
  address="0x123..."
  chain={base}
  className="text-xl font-bold"
>
  <Badge />
</Name>
```

### `<Identity />`
Context provider for identity components.

Props (`IdentityReact`):
```typescript
type IdentityReact = {
  address?: Address;           // The Ethereum address
  chain?: Chain;              // Optional chain for resolution
  children: ReactNode;        // Child components
  className?: string;         // Optional styling
  schemaId?: Address | null;  // Schema for attestation
  hasCopyAddressOnClick?: boolean;
};
```

Example usage:
```typescript
<Identity
  address="0x123..."
  schemaId="0xschema..."
  hasCopyAddressOnClick
>
  <Avatar />
  <Name>
    <Badge />
  </Name>
  <Address />
</Identity>
```

## Wallet Components

### `<Wallet />`
Main container for wallet functionality.

Props (`WalletReact`):
```typescript
type WalletReact = {
  children: ReactNode;
};
```

### `<ConnectWallet />`
Handles wallet connection interface.

Props (`ConnectWalletReact`):
```typescript
type ConnectWalletReact = {
  children?: ReactNode;
  className?: string;
  text?: string;
  withWalletAggregator?: boolean;
};
```

### `<WalletDropdown />`
Dropdown menu for wallet interactions.

Props (`WalletDropdownReact`):
```typescript
type WalletDropdownReact = {
  children: ReactNode;
  className?: string;
};
```

Complete wallet example:
```typescript
<Wallet>
  <ConnectWallet withWalletAggregator>
    <Avatar className="h-6 w-6" />
    <Name />
  </ConnectWallet>
  <WalletDropdown>
    <Identity className="px-4 pt-3 pb-2" hasCopyAddressOnClick>
      <Avatar />
      <Name />
      <Address />
      <EthBalance />
    </Identity>
    <WalletDropdownBasename />
    <WalletDropdownLink
      icon="wallet"
      href="https://example.com"
    >
      Wallet
    </WalletDropdownLink>
    <WalletDropdownDisconnect />
  </WalletDropdown>
</Wallet>
```

## Transaction Components

### `<Transaction />`
Handles complete transaction lifecycle.

Props (`TransactionReact`):
```typescript
type TransactionReact = {
  calls?: Call[] | Promise<Call[]> | (() => Promise<Call[]>);
  capabilities?: WalletCapabilities;
  chainId?: number;
  children: ReactNode;
  className?: string;
  contracts?: ContractFunctionParameters[];
  onError?: (e: TransactionError) => void;
  onStatus?: (lifecycleStatus: LifecycleStatus) => void;
  onSuccess?: (response: TransactionResponse) => void;
};
```

### `<TransactionButton />`
Transaction initiation button.

Props (`TransactionButtonReact`):
```typescript
type TransactionButtonReact = {
  className?: string;
  disabled?: boolean;
  text?: string;
};
```

Complete transaction example:
```typescript
<Transaction
  contracts={[{
    address: contractAddress,
    abi: contractAbi,
    functionName: 'functionName',
    args: [arg1, arg2]
  }]}
  onStatus={(status) => {
    if (status.statusName === 'success') {
      console.log('Transaction successful');
    }
  }}
>
  <TransactionButton text="Submit Transaction" />
  <TransactionStatus>
    <TransactionStatusLabel />
    <TransactionStatusAction />
  </TransactionStatus>
  <TransactionToast />
</Transaction>
```

## Swap Components

### `<Swap />`
Handles token swap functionality.

Props (`SwapReact`):
```typescript
type SwapReact = {
  children: ReactNode;
  className?: string;
  config?: SwapConfig;
  experimental?: {
    useAggregator: boolean;
  };
  isSponsored?: boolean;
  onError?: (error: SwapError) => void;
  onStatus?: (lifecycleStatus: LifecycleStatus) => void;
  onSuccess?: (receipt: TransactionReceipt) => void;
  title?: string;
};
```

### `<SwapAmountInput />`
Input field for swap amounts.

Props (`SwapAmountInputReact`):
```typescript
type SwapAmountInputReact = {
  className?: string;
  delayMs?: number;
  label: string;
  swappableTokens?: Token[];
  token?: Token;
  type: 'to' | 'from';
};
```

Complete swap example:
```typescript
<Swap
  isSponsored={true}
  onStatus={(status) => {
    if (status.statusName === 'success') {
      console.log('Swap successful');
    }
  }}
>
  <SwapAmountInput
    label="Sell"
    swappableTokens={tokens}
    token={fromToken}
    type="from"
  />
  <SwapToggleButton />
  <SwapAmountInput
    label="Buy"
    swappableTokens={tokens}
    token={toToken}
    type="to"
  />
```

~~~
  <SwapButton />
  <SwapMessage />
  <SwapToast />
</Swap>
```

## Fund Components

### `<FundButton />`
Provides access to funding options.

Props (`FundButtonReact`):
```typescript
type FundButtonReact = {
  className?: string;
  disabled?: boolean;
  text?: string;
  hideText?: boolean;
  hideIcon?: boolean;
  fundingUrl?: string;
  openIn?: 'popup' | 'tab';
  popupSize?: 'sm' | 'md' | 'lg';
  rel?: string;
  target?: string;
};
```

Example usage:
```typescript
// Basic usage
<FundButton />

// Customized
<FundButton
  text="Add Funds"
  openIn="popup"
  popupSize="lg"
  fundingUrl={customUrl}
  className="bg-blue-500 text-white"
/>
```

## Frame Components

### `<FrameMetadata />`
Handles Frame metadata for social platforms.

Props (`FrameMetadataReact`):
```typescript
type FrameMetadataReact = FrameMetadataType & {
  ogDescription?: string;
  ogTitle?: string;
  wrapper?: React.ComponentType<any>;
};
```

Example usage:
```typescript
~~~

~~~
<FrameMetadata
  buttons={[
    { label: 'Action 1' },
    {
      action: 'link',
      label: 'Visit Site',
      target: 'https://example.com'
    }
  ]}
  image={{
    src: 'https://example.com/image.png',
    aspectRatio: '1:1'
  }}
  input={{
    text: 'Enter text...'
  }}
  postUrl="https://api.example.com/frame"
/>
```

## Component Combinations

### Identity with Wallet
```typescript
<Identity address="0x123..." hasCopyAddressOnClick>
  <Wallet>
    <ConnectWallet>
      <Avatar className="h-6 w-6" />
      <Name />
    </ConnectWallet>
    <WalletDropdown>
      <Avatar />
      <Name>
        <Badge />
      </Name>
      <Address />
      <EthBalance />
      <WalletDropdownDisconnect />
    </WalletDropdown>
  </Wallet>
</Identity>
```

### Transaction with Swap
```typescript
<Transaction>
  <Swap>
    <SwapAmountInput type="from" token={tokenA} />
    <SwapToggleButton />
    <SwapAmountInput type="to" token={tokenB} />
    <SwapButton />
    <SwapMessage />
    <TransactionStatus />
  </Swap>
</Transaction>
```

### Common State Management Patterns
~~~

```typescript
// Lifecycle status handling
const handleStatus = (status: LifecycleStatus) => {
  switch (status.statusName) {
    case 'init':
      // Handle initialization
      break;
    case 'success':
      // Handle success
      break;
    case 'error':
      // Handle error
      break;
    default:
      // Handle other states
  }
};

// Using in components
<Transaction onStatus={handleStatus}>
  {/* Component children */}
</Transaction>
```

# OnchainKit Checkout Component Reference

## Core Components

### `<Checkout />`
Main container for checkout functionality. Handles payment processing and transaction lifecycle.

Props (`CheckoutReact`):
```typescript
type CheckoutReact = {
  chargeHandler?: () => Promise<string>;  // Custom charge creation handler
  children: React.ReactNode;              // Child components
  className?: string;                     // Optional styling
  isSponsored?: boolean;                  // Enable gas sponsorship
  onStatus?: (status: LifecycleStatus) => void;  // Status callback
  productId?: string;                     // Coinbase Commerce product ID
};
```

### `<CheckoutButton />`
Initiates the checkout process.

Props (`CheckoutButtonReact`):
```typescript
type CheckoutButtonReact = {
  className?: string;          // Optional styling
  coinbaseBranded?: boolean;   // Show Coinbase branding
  disabled?: boolean;          // Disable button
  icon?: React.ReactNode;      // Custom icon
  text?: string;               // Button text
};
```

### `<CheckoutStatus />`
Displays current checkout status.

Props (`CheckoutStatusReact`):
```typescript
type CheckoutStatusReact = {
  className?: string;  // Optional styling
};
```

## Basic Implementation

```typescript
import {
  Checkout,
  CheckoutButton,
  CheckoutStatus
} from '@coinbase/onchainkit/checkout';

// Simple implementation with product ID
<Checkout productId="your-product-id">
  <CheckoutButton />
  <CheckoutStatus />
</Checkout>

// With Coinbase branding and custom text
<Checkout productId="your-product-id">
  <CheckoutButton
    coinbaseBranded
    text="Pay with Crypto"
  />
  <CheckoutStatus />
</Checkout>
```

## Advanced Implementation

### With Status Handling
```typescript
import { Checkout, CheckoutButton } from '@coinbase/onchainkit/checkout';
import type { LifecycleStatus } from '@coinbase/onchainkit/checkout';

function CheckoutComponent() {
  const handleStatus = async (status: LifecycleStatus) => {
    const { statusName, statusData } = status;

    switch (statusName) {
      case 'success':
        const { chargeId, transactionReceipt, receiptUrl } = statusData;
        // Handle successful payment
        break;

      case 'error':
        // Handle error
        break;

      case 'pending':
        // Handle pending state
```

~~~
        break;
    }
  };

  return (
    <Checkout
      productId="your-product-id"
      onStatus={handleStatus}
      isSponsored={true}
    >
      <CheckoutButton coinbaseBranded />
      <CheckoutStatus />
    </Checkout>
  );
}
```

### Custom Charge Handler (Shopping Cart)
```typescript
import { Checkout, CheckoutButton } from '@coinbase/onchainkit/checkout';

function ShoppingCartCheckout() {
  const createCharge = async () => {
    // Create charge on your backend
    const response = await fetch('api/createCharge', {
      method: 'POST',
      body: JSON.stringify({
        // Cart details
        items: cart.items,
        total: cart.total
      })
    });

    const data = await response.json();
    return data.chargeId;
  };

  return (
    <Checkout chargeHandler={createCharge}>
      <CheckoutButton text="Complete Purchase" />
      <CheckoutStatus />
    </Checkout>
  );
}
```

## Lifecycle Status Types

```typescript
type LifecycleStatus =
  | {
      statusName: 'init';
      statusData: LifecycleStatusDataShared;
    }
  | {
      statusName: 'error';
      statusData: TransactionError;
    }
~~~

~~~
  | {
      statusName: 'fetchingData';
      statusData: LifecycleStatusDataShared;
    }
  | {
      statusName: 'ready';
      statusData: {
        chargeId: string;
        contracts: ContractFunctionParameters[];
      };
    }
  | {
      statusName: 'pending';
      statusData: LifecycleStatusDataShared;
    }
  | {
      statusName: 'success';
      statusData: {
        transactionReceipts: TransactionReceipt[];
        chargeId: string;
        receiptUrl: string;
      };
    };
```

## Style Customization

### Basic Styling
```typescript
<Checkout productId="your-product-id">
  <CheckoutButton
    className="bg-blue-500 hover:bg-blue-600 text-white font-bold py-2 px-4
rounded"
  />
  <CheckoutStatus
    className="mt-4 text-sm text-gray-600"
  />
</Checkout>
```

### With Themed Components
```typescript
<OnchainKitProvider
  config={{
    appearance: {
      theme: 'custom',
      mode: 'dark'
    }
  }}
>
  <Checkout productId="your-product-id">
    <CheckoutButton coinbaseBranded />
    <CheckoutStatus />
  </Checkout>
</OnchainKitProvider>
```

## Best Practices
~~~

1. **Error Handling**
```typescript
<Checkout
  productId="your-product-id"
  onStatus={(status) => {
    if (status.statusName === 'error') {
      const { code, message } = status.statusData;
      // Handle specific error cases
      switch (code) {
        case 'INSUFFICIENT_FUNDS':
          notifyUser('Insufficient funds for purchase');
          break;
        // Handle other error cases
      }
    }
  }}
>
  <CheckoutButton />
  <CheckoutStatus />
</Checkout>
```

2. **Transaction Verification**
```typescript
const verifyCharge = async (chargeId: string) => {
  const response = await
fetch(`https://api.commerce.coinbase.com/charges/${chargeId}`, {
    headers: {
      'X-CC-Api-Key': 'your_api_key',
      'Content-Type': 'application/json'
    }
  });
  return response.json();
};

<Checkout
  productId="your-product-id"
  onStatus={async (status) => {
    if (status.statusName === 'success') {
      const { chargeId } = status.statusData;
      const verification = await verifyCharge(chargeId);
      // Handle verification result
    }
  }}
>
  <CheckoutButton />
  <CheckoutStatus />
</Checkout>
```

3. **Custom Button States**
```typescript
<Checkout productId="your-product-id">
  <CheckoutButton
    text={isLoading ? 'Processing...' : 'Complete Purchase'}
    disabled={isLoading || !isValid}
    className={`
```

```
      ${isLoading ? 'opacity-50 cursor-not-allowed' : ''}
      ${!isValid ? 'bg-gray-300' : 'bg-blue-500 hover:bg-blue-600'}
    `}
  />
  <CheckoutStatus />
</Checkout>
```