Plugin Directory

Changeset 3480315 for woocommerce-payments/trunk/includes/class-wc-payment-gateway-wcpay.php

Timestamp: 03/11/2026 03:29:25 PM (3 weeks ago)
Author: automattic

Message: Tagging version 10.6.0

File: 1 edited

woocommerce-payments/trunk/includes/class-wc-payment-gateway-wcpay.php (19 diffs)

Unmodified Added Removed

woocommerce-payments/trunk/includes/class-wc-payment-gateway-wcpay.php — Tabular | Unified

r3454336r3480315

```
38  38  use WCPay\Core\Server\Request\Create_And_Confirm_Intention;
39  39  use WCPay\Core\Server\Request\Create_And_Confirm_Setup_Intention;
    40  use WCPay\Core\Server\Request\Create_Setup_Intention;
40  41  use WCPay\Core\Server\Request\Get_Charge;
41  42  use WCPay\Core\Server\Request\Get_Intention;
...
112 113     const USER_FORMATTED_TOKENS_LIMIT = 100;
113 114
114     const PROCESS_REDIRECT_ORDER_MISMATCH_ERROR_CODE        =
'upe_process_redirect_order_id_mismatched';
115     const UPE_APPEARANCE_TRANSIENT                          = 'wcpay_upe_appearance';
116     const UPE_ADD_PAYMENT_METHOD_APPEARANCE_TRANSIENT       =
'wcpay_upe_add_payment_method_appearance';
117     const WC_BLOCKS_UPE_APPEARANCE_TRANSIENT                = 'wcpay_wc_blocks_upe_appearance';
118     const UPE_BNPL_PRODUCT_PAGE_APPEARANCE_TRANSIENT        =
'wcpay_upe_bnpl_product_page_appearance';
119     const UPE_BNPL_CLASSIC_CART_APPEARANCE_TRANSIENT        =
'wcpay_upe_bnpl_classic_cart_appearance';
120     const UPE_BNPL_CART_BLOCK_APPEARANCE_TRANSIENT          =
'wcpay_upe_bnpl_cart_block_appearance';
121     const UPE_APPEARANCE_THEME_TRANSIENT                    = 'wcpay_upe_appearance_theme';
122     const UPE_ADD_PAYMENT_METHOD_APPEARANCE_THEME_TRANSIENT =
'wcpay_upe_add_payment_method_appearance_theme';
123     const WC_BLOCKS_UPE_APPEARANCE_THEME_TRANSIENT          =
'wcpay_wc_blocks_upe_appearance_theme';
124     const UPE_BNPL_PRODUCT_PAGE_APPEARANCE_THEME_TRANSIENT  =
'wcpay_upe_bnpl_product_page_appearance_theme';
125     const UPE_BNPL_CLASSIC_CART_APPEARANCE_THEME_TRANSIENT  =
'wcpay_upe_bnpl_classic_cart_appearance_theme';
126     const UPE_BNPL_CART_BLOCK_APPEARANCE_THEME_TRANSIENT    =
'wcpay_upe_bnpl_cart_block_appearance_theme';
127
128     /**
129      * The locations of appearance transients.
130      */
131     const APPEARANCE_THEME_TRANSIENTS = [
132         'checkout'     => [
133             'blocks'  => self::WC_BLOCKS_UPE_APPEARANCE_THEME_TRANSIENT,
```

r3454336 r3480315

```
            'classic' => self::UPE_APPEARANCE_THEME_TRANSIENT,
        ],
        'product_page' => [
            'blocks'  => self::UPE_BNPL_PRODUCT_PAGE_APPEARANCE_THEME_TRANSIENT,
            'classic' => self::UPE_BNPL_PRODUCT_PAGE_APPEARANCE_THEME_TRANSIENT,
        ],
        'cart'         => [
            'blocks'  => self::UPE_BNPL_CART_BLOCK_APPEARANCE_THEME_TRANSIENT,
            'classic' => self::UPE_BNPL_CLASSIC_CART_APPEARANCE_THEME_TRANSIENT,
        ],
    ];
    const PROCESS_REDIRECT_ORDER_MISMATCH_ERROR_CODE = 'upe_process_redirect_order_id_mismatched';

    /**
...
        if ( $this->is_saved_cards_enabled() ) {
            array_push( $this->supports, 'tokenization', 'add_payment_method' );
        }

        // enabling the custom place order button for express checkout methods (Apple Pay, Google Pay, Amazon Pay)
        // only when the feature is available. Other payment methods like WooPay or card will return `false` for `is_express_checkout()`.
        if ( property_exists( $this, 'has_custom_place_order_button' ) && $this->payment_method->is_express_checkout() && \WC_Payments::get_gateway()->is_express_checkout_in_payment_methods_enabled() ) {
            $this->has_custom_place_order_button = true;
            $this->has_fields                    = false;
        }
    }
...
     */
    public function is_available() {
        // Express checkout methods (Apple Pay, Google Pay, Amazon Pay) are only available
        // in the payment methods list when the feature is enabled. Otherwise, they appear
        // as separate express checkout buttons.
        if ( $this->payment_method->is_express_checkout() && ! is_admin() ) {
            if ( ! WC_Payments::get_gateway()->is_express_checkout_in_payment_methods_enabled() ) {
                return false;
            }
        }

        return $this->check_base_availability();
    }

    /**
     * Checks base availability without checkout-page-specific restrictions.
     * Used by is_available_for_express_checkout() for payment methods that are
     * only available via express checkout (e.g., Amazon Pay).
     *
     * @return bool
     */
    protected function check_base_availability() {
        if ( ! WC_Payments::get_gateway()->is_enabled() ) {
            return false;
...

        // Disable the gateway if it should not be displayed on the checkout page.
        $is_gateway_enabled = in_array( $this->stripe_id, $this->get_payment_method_ids_enabled_at_checkout(), true ) ? true : false;
        $is_gateway_enabled = in_array( $this->stripe_id, $this->get_payment_method_ids_enabled_at_checkout(), true );
```

r3454336 r3480315

```
929 927         if ( ! $is_gateway_enabled ) {
930 928             return false;
... ...
932 930
933 931         return parent::is_available() && ! $this->needs_setup();
    932     }
    933
    934     /**
    935      * Checks if the gateway is available for express checkout.
    936      * This bypasses checkout-page-specific restrictions for payment methods
    937      * that are only available via express checkout buttons.
    938      *
    939      * @return bool
    940      */
    941     public function is_available_for_express_checkout() {
    942         if ( is_admin() ) {
    943             // In admin context (e.g. block editor preview), skip full availability
    944             // checks. check_base_availability() includes runtime checks (HTTPS,
    945             // currency, capability status) that can fail without an active cart
    946             // or customer session. A simple enabled check is sufficient here.
    947             return WC_Payments::get_gateway()->is_enabled() && $this->is_enabled();
    948         }
    949
    950         return $this->check_base_availability();
934 951     }
935 952
... ...
975 992
976 993         return false;
    994     }
    995
    996     /**
    997      * Whether express checkout methods should appear in the payment methods list
    998      * instead of as separate express buttons.
    999      *
   1000      * Requires both the dynamic checkout place order button feature flag
   1001      * and the express_checkout_in_payment_methods gateway setting.
   1002      *
   1003      * @return bool
   1004      */
   1005     public function is_express_checkout_in_payment_methods_enabled(): bool {
   1006         return WC_Payments_Features::is_dynamic_checkout_place_order_button_enabled()
   1007             && 'yes' === $this->get_option( 'express_checkout_in_payment_methods' );
977 1008     }
978 1009
... ...
1717 1748
1718 1749                 // For $0 orders, we need to save the payment method using a setup intent.
1719                     $request = Create_And_Confirm_Setup_Intention::create();
1720                     $request->set_customer( $customer_id );
1721
1722                     // Setting the credential based on what was provided.
1723 1750                 $payment_credential = $payment_information->get_payment_method();
     1751
     1752                 // For confirmation tokens (e.g.: through the ECE), we must create an unconfirmed `SetupIntent`
     1753                 // and let the frontend confirm it with the confirmation token.
     1754                 // Stripe's SetupIntent API doesn't support confirmation_token with confirm=true in the same way `PaymentIntent`s do.
1724 1755                 if ( $payment_information->is_using_confirmation_token() ) {
1725                         $request->set_confirmation_token( $payment_credential );
     1756                         $request = Create_Setup_Intention::create();
```

r3454336 r3480315

```
                    $request->set_customer( $customer_id );
                    $request->set_payment_method_types( $this->get_payment_method_types( $payment_information ) );
                    $request->set_metadata( $metadata );
                    $request->assign_hook( 'wcpay_create_setup_intention_request' );
                } else {
                    $request = Create_And_Confirm_Setup_Intention::create();
                    $request->set_customer( $customer_id );
                    $request->set_payment_method( $payment_credential );
                }
                $request->set_metadata( $metadata );
                $request->assign_hook( 'wcpay_create_and_confirm_setup_intention_request' );
                $request->set_hook_args( $payment_information, false, $save_user_in_woopay );

                if (
                    Payment_Method::CARD === $this->get_selected_stripe_payment_type_id() &&
                    in_array( Payment_Method::LINK, $this->get_upe_enabled_payment_method_ids(), true )
                    $request->set_metadata( $metadata );
                    $request->assign_hook( 'wcpay_create_and_confirm_setup_intention_request' );
                    $request->set_hook_args( $payment_information, false, $save_user_in_woopay );

                    if (
                        Payment_Method::CARD === $this->get_selected_stripe_payment_type_id() &&
                        in_array( Payment_Method::LINK, $this->get_upe_enabled_payment_method_ids(), true )
                    ) {
                    $request->set_payment_method_types( $this->get_payment_method_types( $payment_information ) );
                    $request->set_mandate_data( $this->get_mandate_data() );
                        $request->set_payment_method_types( $this->get_payment_method_types( $payment_information ) );
                        $request->set_mandate_data( $this->get_mandate_data() );
                    }
                }


            }

            if ( Intent_Status::REQUIRES_ACTION === $status ) {
            $needs_frontend_confirmation = (
                Intent_Status::REQUIRES_ACTION === $status
                || Intent_Status::REQUIRES_CONFIRMATION === $status
                || (
                    // For SetupIntents with confirmation tokens, the status will be 'requires_payment_method'
                    // since no payment method is attached yet (the confirmation token will be used on frontend).
                    Intent_Status::REQUIRES_PAYMENT_METHOD === $status
                    && $payment_information->is_using_confirmation_token()
                    && ! $payment_needed
                )
            );

            if ( $needs_frontend_confirmation ) {
                $next_action_type = $next_action['type'] ?? null;
                if ( 'redirect_to_url' === $next_action_type && ! empty( $next_action[ $next_action_type ]['url'] ) ) {
                    );
                } else {
                    // Build the redirect URL with the confirmation token for `SetupIntent`s requested through the ECE.
```

r3454336 r3480315

```
                    // Format: #wcpay-confirm-{si|pi}:{orderId}:{clientSecret}:{nonce}[:{confirmationToken}].
                    $redirect_hash_parts = [
                        $payment_needed ? 'pi' : 'si',
                        $order_id,
                        $client_secret,
                        wp_create_nonce( 'wcpay_update_order_status_nonce' ),
                    ];

                    // For ECE SetupIntents, include the confirmation token so the frontend can
                    // use it with confirmSetup() to complete the confirmation.
                    if ( ! $payment_needed && $payment_information->is_using_confirmation_token() ) {
                        $redirect_hash_parts[] = $payment_information->get_payment_method();
                    }

                    $response = [
                        'result'         => 'success',
                        // Include a new nonce for update_order_status to ensure the update order
                        // status call works when a guest user creates an account during checkout.
                        'redirect'       => sprintf(
                            '#wcpay-confirm-%s:%s:%s:%s',
                            $payment_needed ? 'pi' : 'si',
                            $order_id,
                            $client_secret,
                            wp_create_nonce( 'wcpay_update_order_status_nonce' ),
                        ),
                        'redirect'       => '#wcpay-confirm-' . implode( ':', $redirect_hash_parts ),
                        // Include the payment method ID so the Blocks integration can save cards.
                        'payment_method' => $payment_information->get_payment_method(),
...
        $this->set_payment_method_title_for_order( $order, $payment_method_type, $payment_method_details );

        if ( isset( $status ) && Intent_Status::REQUIRES_ACTION === $status && $this->is_changing_payment_method_for_subscription() ) {
        if ( isset( $status ) && ( Intent_Status::REQUIRES_ACTION === $status || Intent_Status::REQUIRES_CONFIRMATION === $status ) && $this->is_changing_payment_method_for_subscription() ) {
            // Because we're filtering woocommerce_subscriptions_update_payment_via_pay_shortcode, we need to manually set this delayed update all flag here.
            if ( isset( $_POST['update_all_subscriptions_payment_method'] ) && wc_clean( wp_unslash( $_POST['update_all_subscriptions_payment_method'] ) ) ) { // phpcs:ignore WordPress.Security.NonceVerification.Missing
...
        // Otherwise, $gateway_id must be `woocommerce_payments`.
        if ( substr( $gateway_id, 0, strlen( $split_upe_gateway_prefix ) ) === $split_upe_gateway_prefix ) {
            return [ str_replace( $split_upe_gateway_prefix, '', $gateway_id ) ];
            $payment_method = str_replace( $split_upe_gateway_prefix, '', $gateway_id );

            // Apple Pay and Google Pay are wrappers around card payments for Stripe.
            $card_wrappers = [ Payment_Method::APPLE_PAY, Payment_Method::GOOGLE_PAY ];
            if ( in_array( $payment_method, $card_wrappers, true ) ) {
                return [ Payment_Method::CARD ];
            }

            return [ $payment_method ];
        }
...
    public function init_settings() {
        parent::init_settings();
```

r3454336 r3480315

```
2675        $this->enabled = ! empty( $this->settings[ static::METHOD_ENABLED_KEY ] ) && 'yes' === $this->settings[ static::METHOD_ENABLED_KEY ] ? 'yes' : 'no';
2741
2742        // Get the basic enabled value from settings.
2743        $is_enabled = ! empty( $this->settings[ static::METHOD_ENABLED_KEY ] ) && 'yes' === $this->settings[ static::METHOD_ENABLED_KEY ];
2744
2745        // Card and express checkout methods are not in the UPE enabled list,
2746        // so they only need the basic enabled setting check. Without this
2747        // early return, they would fall through to the UPE list verification
2748        // below and always end up disabled.
2749        if ( 'card' === $this->stripe_id || $this->payment_method->is_express_checkout() ) {
2750            return;
2751        }
2752
2753        // For split gateways, also verify the method is in the UPE enabled list.
2754        // This prevents sync issues where a gateway has enabled=yes but isn't
2755        // actually configured for checkout in the UPE settings.
2756        if ( $is_enabled ) {
2757            $upe_enabled_methods = $this->get_upe_enabled_payment_method_ids();
2758            $this->enabled       = in_array( $this->stripe_id, $upe_enabled_methods, true ) ? 'yes' : 'no';
2759        } else {
2760            $this->enabled = 'no';
2761        }
2676 2762    }
2677 2763
...  ...
3683 3769                $setup_intent_request = Get_Setup_Intention::create( $intent_id );
3684 3770                /** @var WC_Payments_API_Setup_Intention $setup_intent */ // phpcs:ignore Generic.Commenting.DocComment.MissingShort
3685                     $intent    = $setup_intent_request->send();
3686                     $status    = $intent->get_status();
3687                     $charge_id = '';
3771                $intent = $setup_intent_request->send();
3772                $status = $intent->get_status();
3773
3774                // For $0 orders (free trials), directly complete the order when SetupIntent succeeds.
3775                // This is similar to how WC Stripe Gateway handles it - calling payment_complete()
3776                // directly ensures the order transitions to the correct status and activates subscriptions.
3777                // Otherwise, the order would be in a "Pending payment" state and the subscription would be "Pending".
3778                if ( Intent_Status::SUCCEEDED === $status && ! $order->is_paid() ) {
3779                    $order->payment_complete( $intent_id );
3780
3781                    // Add a success note similar to mark_payment_completed().
3782                    $note = sprintf(
3783                    /* translators: %1: the successfully charged amount, %2: WooPayments, %3: transaction ID of the payment */
3784                        __( 'A payment of %1$s was successfully charged using %2$s (%3$s).', 'woocommerce-payments' ),
3785                        wc_price( $order->get_total(), [ 'currency' => $order->get_currency() ] ),
3786                        'WooPayments',
3787                        $intent_id
3788                    );
3789                    $order->add_order_note( $note );
3790                    $this->order_service->set_intention_status_for_order( $order, $status );
3791                    $order->save();
3792                }
3688 3793            }
3689 3794
```

r3454336 r3480315

```
3690 3795             $payment_method_id = $intent->get_payment_method_id();
     3796
     3797             // For SetupIntents confirmed via frontend (e.g., ECE with confirmation tokens),
     3798             // store the payment method ID in order meta. This ensures subscription renewals
     3799             // can find the payment method even if token creation fails later.
     3800             if ( ! empty( $payment_method_id ) ) {
     3801                 $this->order_service->set_payment_method_id_for_order( $order, $payment_method_id );
     3802             }
3691 3803
3692 3804             if ( Intent_Status::SUCCEEDED === $status ) {
...  ...
3711 3823                         }
3712 3824                     } catch ( Exception $e ) {
3713                          // If saving the token fails, log the error message but catch the error to avoid crashing the checkout flow.
3714 3825                         Logger::log( 'Error when saving payment method: ' . $e->getMessage() );
     3826
     3827                         // For subscription orders, token creation failure is critical - renewals will fail.
     3828                         // Re-throw the exception so the customer sees an error instead of a successful
     3829                         // checkout that will fail on the first renewal.
     3830                         if ( $is_subscription ) {
     3831                             throw new Exception(
     3832                                 __( 'Unable to save payment method for subscription. Please try again or use a different payment method.', 'woocommerce-payments' )
     3833                             );
     3834                         }
3715 3835                     }
3716 3836                 }
...  ...
3966 4086     * Returns a formatted token list for a user.
3967 4087     *
3968          * @param int $user_id The user ID.
3969          */
3970         protected function get_user_formatted_tokens_array( $user_id ) {
     4088     * @param int         $user_id The user ID.
     4089     * @param string|null $gateway_id Optional gateway ID to filter tokens. Defaults to card gateway.
     4090     */
     4091     protected function get_user_formatted_tokens_array( $user_id, $gateway_id = null ) {
3971 4092         $tokens = WC_Payment_Tokens::get_tokens(
3972 4093             [
3973 4094                 'user_id'    => $user_id,
3974                      'gateway_id' => self::GATEWAY_ID,
     4095                 'gateway_id' => $gateway_id ?? self::GATEWAY_ID,
3975 4096                 'limit'      => self::USER_FORMATTED_TOKENS_LIMIT,
3976 4097             ]
...  ...
3979 4100         return array_map(
3980 4101             static function ( WC_Payment_Token $token ): array {
     4102                 // ensures that Google Pay/Apple Pay methods display "Google Pay Visa ending in 1234",
     4103                 // instead of just "Visa ending in 1234".
     4104                 $wallet_type    = $token->get_meta( '_wcpay_wallet_type', true );
     4105                 $name           = $token->get_display_name();
     4106                 $payment_method = WC_Payments::get_payment_method_by_id( $wallet_type );
     4107                 if ( $payment_method && method_exists( $payment_method, 'get_title' ) ) {
     4108                     $name = join( ' ', [ $payment_method->get_title(), $name ] );
     4109                 }
     4110
3981 4111                 return [
```

r3454336 r3480315

```
3982 4112                    'tokenId'         => $token->get_id(),
3983 4113                    'paymentMethodId' => $token->get_token(),
3984 4114                    'isDefault'       => $token->get_is_default(),
3985                         'displayName'     => $token->get_display_name(),
     4115                    'displayName'     => $name,
3986 4116                ];
3987 4117            },
...  ...
4216 4346    }
4217 4347
4218         /**
4219          * Handle AJAX request for saving UPE appearance value to transient.
4220          *
4221          * @throws Exception - If nonce or setup intent is invalid.
4222          */
4223         public function save_upe_appearance_ajax() {
4224             try {
4225                 $is_nonce_valid = check_ajax_referer( 'wcpay_save_upe_appearance_nonce', false, false );
4226                 if ( ! $is_nonce_valid ) {
4227                     throw new Exception(
4228                         __( 'Unable to update UPE appearance values at this time.', 'woocommerce-payments' )
4229                     );
4230                 }
4231
4232                 $elements_location = isset( $_POST['elements_location'] ) ? wc_clean( wp_unslash( $_POST['elements_location'] ) ) : null;
4233                 $appearance        = isset( $_POST['appearance'] ) ? json_decode( wc_clean( wp_unslash( $_POST['appearance'] ) ) ) : null;
4234
4235                 $valid_locations = [ 'blocks_checkout', 'shortcode_checkout', 'bnpl_product_page', 'bnpl_classic_cart', 'bnpl_cart_block', 'add_payment_method' ];
4236                 if ( ! $elements_location || ! in_array( $elements_location, $valid_locations, true ) ) {
4237                     throw new Exception(
4238                         __( 'Unable to update UPE appearance values at this time.', 'woocommerce-payments' )
4239                     );
4240                 }
4241
4242                 if ( in_array( $elements_location, [ 'blocks_checkout', 'shortcode_checkout' ], true ) ) {
4243                     $is_blocks_checkout = 'blocks_checkout' === $elements_location;
4244                     /**
4245                      * This filter is only called on "save" of the appearance, to avoid calling it on every page load.
4246                      * If you apply changes through this filter, you'll need to clear the transient data to see them at checkout.
4247                      *
4248                      * @deprecated 7.4.0 Use {@see 'wcpay_elements_appearance'} instead.
4249                      * @since 7.3.0
4250                      */
4251                     $appearance = apply_filters_deprecated( 'wcpay_upe_appearance', [ $appearance, $is_blocks_checkout ], '7.4.0', 'wcpay_elements_appearance' );
4252                 }
4253
4254                 /**
4255                  * This filter is only called on "save" of the appearance, to avoid calling it on every page load.
4256                  * If you apply changes through this filter, you'll need to clear the transient data to see them at checkout.
4257                  * $elements_location can be 'blocks_checkout', 'shortcode_checkout', 'bnpl_product_page', 'bnpl_classic_cart', 'bnpl_cart_block', 'add_payment_method'.
4258                  *
```

r3454336 r3480315

```
4259            * @since 7.4.0
4260            */
4261           $appearance = apply_filters( 'wcpay_elements_appearance', $appearance, $elements_location );
4262
4263           $appearance_transient       = [
4264               'shortcode_checkout' => self::UPE_APPEARANCE_TRANSIENT,
4265               'add_payment_method' => self::UPE_ADD_PAYMENT_METHOD_APPEARANCE_TRANSIENT,
4266               'blocks_checkout'    => self::WC_BLOCKS_UPE_APPEARANCE_TRANSIENT,
4267               'bnpl_product_page'  => self::UPE_BNPL_PRODUCT_PAGE_APPEARANCE_TRANSIENT,
4268               'bnpl_classic_cart'  => self::UPE_BNPL_CLASSIC_CART_APPEARANCE_TRANSIENT,
4269               'bnpl_cart_block'    => self::UPE_BNPL_CART_BLOCK_APPEARANCE_TRANSIENT,
4270           ][ $elements_location ];
4271           $appearance_theme_transient = [
4272               'shortcode_checkout' => self::UPE_APPEARANCE_THEME_TRANSIENT,
4273               'add_payment_method' => self::UPE_ADD_PAYMENT_METHOD_APPEARANCE_THEME_TRANSIENT,
4274               'blocks_checkout'    => self::WC_BLOCKS_UPE_APPEARANCE_THEME_TRANSIENT,
4275               'bnpl_product_page'  => self::UPE_BNPL_PRODUCT_PAGE_APPEARANCE_THEME_TRANSIENT,
4276               'bnpl_classic_cart'  => self::UPE_BNPL_CLASSIC_CART_APPEARANCE_THEME_TRANSIENT,
4277               'bnpl_cart_block'    => self::UPE_BNPL_CART_BLOCK_APPEARANCE_THEME_TRANSIENT,
4278           ][ $elements_location ];
4279
4280           if ( null !== $appearance ) {
4281               set_transient( $appearance_transient, $appearance, DAY_IN_SECONDS );
4282               set_transient( $appearance_theme_transient, $appearance->theme, DAY_IN_SECONDS );
4283           }
4284
4285           wp_send_json_success( $appearance, 200 );
4286       } catch ( Exception $e ) {
4287           // Send back error so it can be displayed to the customer.
4288           wp_send_json_error(
4289               [
4290                   'error' => [
4291                       'message' => WC_Payments_Utils::get_filtered_error_message( $e ),
4292                   ],
4293               ],
4294               WC_Payments_Utils::get_filtered_error_status_code( $e )
4295           );
4296       }
4297   }
4298
4299   /**
4300    * Clear the saved UPE appearance transient value.
4301    */
4302   public function clear_upe_appearance_transient() {
4303       delete_transient( self::UPE_APPEARANCE_TRANSIENT );
4304       delete_transient( self::WC_BLOCKS_UPE_APPEARANCE_TRANSIENT );
4305       delete_transient( self::UPE_BNPL_PRODUCT_PAGE_APPEARANCE_TRANSIENT );
4306       delete_transient( self::UPE_BNPL_CLASSIC_CART_APPEARANCE_TRANSIENT );
4307       delete_transient( self::UPE_BNPL_CART_BLOCK_APPEARANCE_TRANSIENT );
4308       delete_transient( self::UPE_APPEARANCE_THEME_TRANSIENT );
4309       delete_transient( self::WC_BLOCKS_UPE_APPEARANCE_THEME_TRANSIENT );
4310       delete_transient( self::UPE_BNPL_PRODUCT_PAGE_APPEARANCE_THEME_TRANSIENT );
4311       delete_transient( self::UPE_BNPL_CLASSIC_CART_APPEARANCE_THEME_TRANSIENT );
4312       delete_transient( self::UPE_BNPL_CART_BLOCK_APPEARANCE_THEME_TRANSIENT );
4313   }
      4348
4314  4349
4315  4350   /**
...   ...
4487  4522
```

r3454336 r3480315

```
4488 4523     /**
4489          * Checks if UPE appearance theme is set and returns appropriate icon URL.
     4524     * Returns the appropriate icon URL for the payment method.
4490 4525     *
4491 4526     * @return string
4492 4527     */
4493 4528    public function get_theme_icon() {
4494              $upe_appearance_theme = get_transient( self::UPE_APPEARANCE_THEME_TRANSIENT );
4495              if ( $upe_appearance_theme ) {
4496                  return 'night' === $upe_appearance_theme ? $this->payment_method->get_dark_icon() : $this->payment_method->get_icon();
4497              }
4498 4529         return $this->payment_method->get_icon();
4499 4530     }
```