

---

# ModernBill v5 Developers Guide

*All Articles in All Categories*

---

# Contents

<b>Directory Structure</b>	1
<i>ModernBill Directory Structure</i>	1
<b>Action API</b>	2
<i>API Action Introduction</i>	2
<b>Protocol</b>	2
<i>Action API Protocol</i>	2
<b>Error Codes</b>	3
<i>Action API Error Codes</i>	3
<b>MBAPI</b>	3
<b>Writing Modules</b>	3
<b>Panel Shell</b>	4
<b>MBAPI Commands &amp; Examples</b>	4
<i>MBAPI Client Authentication</i>	4
<b>ProcessCheckout</b>	6
<i>How To Inject An Order Into ModernBill Using ProcessCheckout</i>	6
<b>Action Hooks</b>	11
<i>Writing Action Hooks</i>	11
<b>Helpdesk Integration Example</b>	12
<i>Using Action Hooks for 3rd-party Helpdesk Integrations</i>	12

# Directory Structure

A description of the ModernBill directory structure.

## ModernBill Directory Structure

ModernBill is developed using the MBAPI framework. Each distribution will include the ModernBill application directories as well as the supporting MBAPI directories.

app-modernbill5

The primary meta directory that defines ModernBill's dependencies.

app-modernbill-api

The ModernBill Action API Files

app-modernbill-admin

The ModernBill Admin Integration Action Files

app-modernbill-client

The ModernBill Client Integration Action Files

app-modernbill-order

The ModernBill Order Integration Action Files

app-helpdesk

The Generic Helpdesk Integration Action Files

lib-action

The MBAPI Library for the Action Integration Files

lib-api-action

The MBAPI Library for the Action API Files

lib-data

The MBAPI Library for Data Manipulation

lib-db

The MBAPI Library for ADODB and Query master

lib-domain

The MBAPI Library for IDN Domain Names and Whois Services

lib-import

The MBAPI Library for Migrations

lib-mbapi

The Primary Library for the MBAPI Framework

lib-modernbill

The Primary Library for the ModernBill Application

lib-pkg

The MBAPI Library for Building Application Dependencies

lib-server

The MBAPI Library for Running the MBAPI as a Service

lib-themes

The MBAPI Library for all Application Theming

lib-tk

The Secondary Library for the MBAPI Framework (tk = Tool Kit)

sdk  
The MBAPI SDK Files

makeapp  
The MBAPI Library used to build MBAPI Applications for Deployment

## Action API

The Action API is for accessing a small subset of actions remotely.

### API Action Introduction

The Action API (also known as the Business API) is for accessing a subset of ModernBill actions remotely.

Unlike the MBAPI, the Action API can execute one or more MBAPI commands in a single connection!

The following actions are currently supported via the Action API:

- CreateInvoice
- CreatePackage
- GetBillingAccounts
- GetClients
- GetClientStatus
- GetLogEnabledCommands
- GetMBAPIMap
- GetOrderForms
- GetProducts
- UpdatePackage

## Protocol

The connection protocol for the Action API.

### Action API Protocol

The Action API accepts an HTTPS request and returns an XML response.

Example HTTPS Request:  
`https://www.domain.com/app-modernbill-api/api.php?  
apiAction=GetOrderForms&  
orderFormID=1&  
remoteAccessHash=__REMOTE_ACCESS_HASH__`

Example XML Response:

```
app-modernbill-api  
1  
__UNIXTIME__  
  
(...)
```

## Error Codes

The error codes for the Action API.

### Action API Error Codes

If the XML Response returns a success code of 0, then an error has occurred.  
Example XML Response:

0

1000  
Invalid API Action.

Error Codes within the Action API:

1000 Invalid API Action

2000 Supplied client cannot be found

2001 Session did not have client information associated

2002 Username is already taken.

2003 Client supplied differs from session record

2004 Supplied user credentials not found.

3000 Could not validate input

3001 Locale not provided

4000 Invalid language passed with locale.

4001 Failed to record session.

5000 Failed to retrieve domain product information

5001 Failed to retrieve product information

6000 Failed to build invoice.

## MBAPI

The MBAPI is the platform ModernBill was developed on.

*There are no articles in this category.*

## Writing Modules

*There are no articles in this category.*

## Panel Shell

*There are no articles in this category.*

## MBAPI Commands & Examples

### MBAPI Client Authentication

You can use the MBAPI to authenticate a client with their email address or username and their password. See examples below.

Example 1: Authenticate client using Email Address & Password:

```
__REMOTE_ACCESS_HASH__  
GetClients  
1  
  
1  
1  
__PRIMARY_CONTACT_EMAIL__  
__PRIMARY_CONTACT_PASSWORD__
```

Gotcha: The authentication param IS required and MUST be used for email based authentication.

Example 2: Authenticate client using Username & Password:

```
__REMOTE_ACCESS_HASH__  
GetClients  
1  
  
1  
__PRIMARY_CONTACT_USERNAME__  
__PRIMARY_CONTACT_PASSWORD__
```

Gotcha: The authentication param IS NOT required and CAN NOT be used for username based authentication.

A successful MBAPI query will return the following results:

1

1

(...)

(...)

Gotcha: A successful MBAPI query where the success param value is "1" does not mean that the authentication was successful too. You must also make sure that only "1" result was returned in the numResults param.

Note that this method does NOT verify the status of the client or their packages, so if a status changes, the client will still be authenticated.

There are two approaches for dealing with this depending on whether the access is specific to a client's package, or just the client.

#### Method 1: Package access

If the access is specific to a client's package, you will want to:

1. have the access associated with a product or product variant, having the ID of the product or product variant available to the authentication code
2. authenticate the client as shown above, getting their client ID from the results (in the "clientID" element)
3. check for active (and perhaps pending) packages tied to that client and product or product variant.

Checking for active and pending packages:

Request:

```
1
__REMOTE_ACCESS_HASH__
GetData
packages
  __CLIENT_ID__
  __PRODUCT_ID__
  active
  none
  pending
```

Response:

1

1

---

Method 2: Client access

If the access is NOT specific to a client's package, you will want to:

1. authenticate the client as shown above, getting their client ID from the results (in the "clientID" element)
2. verify active (and perhaps pending) status of the client.

Checking for an active or pending client:

Request:

```
1
__REMOTE_ACCESS_HASH__
GetData
clients
  __CLIENT_ID__
  active
  none
  pending
```

Response:

1

1

## ProcessCheckout

### How To Inject An Order Into ModernBill Using ProcessCheckout

The ProcessCheckout command is one of the more powerful MBAPI commands in the system.

---

When used properly, it will do everything necessary to add an order to the system from any local or remote connection. The default ModernBill order shopping cart as well as the single page order form both use this command to do all of their processing (including credit card processing).

The ProcessCheckout command can perform the following actions:

- Create or use an existing client record
- Create or use an existing client contact record
- Create or use an existing billing account
- Optionally run FraudGuardian call
- Return a preview invoice - OR - Create the necessary packages and addons
- Map the specified billing account to pay for those packages
- Create an invoice for those packages
- Create all accounting records for invoice
- Create event queue entries to provision for those packages (ProcessPanel and ProcessRegistrar calls)
- Charge credit card if necessary
- Create the transaction record for the charge
- Create all accounting entries for transaction
- Return the client, FraudGuardian, invoice, and transaction information (including decline and error notices)

Before you can run the ProcessCheckout command, you must already know the following pieces of information:

The orderFormID of the order form you are going to use. This controls currency, locale, domain product selection, checkout options.

The paymentType that you wish to use, valid values are:

- 1 = credit card
- 2 = third party gateway like paypal or worldpay
- 3 = want to pay via check
- 4 = echeck
- 5 = bank draft
- 50 = want to be invoiced
- 9999 = existing billing account

The productID, productVariantID, cycleID, price and setup price, taxZoneGroupID, couponNumCycles, and couponPrice for each product on this order

The crm information for the customer, and separate information for the domain contact info if it is different

Below is a sample of a simple ProcessCheckout call:

```
ProcessCheckout
process

1

1
0
0
0
1

Kris
Bailey
kris@modernbill.com

kriskris
password
21
224
1
```

---

1

123 Some st

city  
KY  
zip  
502-111-2222

0

Kristopher Bailey  
0  
4200000000000000  
03  
2009

123

Kris  
Bailey  
address

city  
KY  
40219  
Company here

kris@modernbill.com  
phone 1  
fax  
224

kriskris  
password

2  
3

12  
1164991718  
3

0

1  
0

10  
0

0  
3

---

3  
1  
  
1  
1164991718  
6  
  
0  
  
1  
0  
  
15  
0  
  
0  
3  
somedomainhere  
com

Example Result (SUCCESS):

1  
0  
  
1164991719  
0

1002  
1002  
252  
1524  
1525  
1004

1  
1  
TRANS\_CHARGES\_SUCCESSFULLY\_APPLIED  
146  
1  
1  
507070203  
This transaction has been approved.  
1164991742  
0  
Y  
25.00  
USD  
77  
1004

---

Example Result (DECLINED TRANSACTION):

```
1
0
1164991719
0

1002
1002
252
1524
1525
1004

1
0
TRANS_SOME_DECLINED_CHARGES
```

In the event that you have a declined transaction, note that the following things are already created in the system:

- Client (if it didn't exist it does now)
- Client Contact (if it didn't exist it does now)
- Billing Account (if it didn't exist it does now)
- Packages
- Invoice
- Event Queue provisioning entries

Gotcha: This means that in order to NOT clutter your system with stale entries, your next ProcessCheckout call for that customer (after you have redisplayed your payment page and asked for corrected payment information) needs to be a little different in that it includes some of these ID's instead of the actual information to create their entries. The system will use the information you have provided previously and only attempt to charge for the services again.

An example of a ProcessCheckout call that uses the information from a previous call is below:

```
ProcessCheckout
process

1
1
0
0
1
252

1002
```

1002

1004  
0

Kristopher Bailey  
0  
40000000000000001  
03  
2009

123

The system will take the invoiceID, clientID, and clientContactID and use them instead of creating new records. It will also take the billingAccountID that you pass it and update it with the information you provide to correct the one that got declined to the new information.

Running that will return the same results (either SUCCESS or DECLINED) as the original command without creating additional clients contacts or invoices.

## Action Hooks

Action hooks allow you to override or integrate with any action in the Action Integration Layer.

### Writing Action Hooks

Step 1: Locate the original action file you wish to override.

Learn about the ModernBill application directory structure here:

<http://manual.modernbill.com/v5dev/index.php?article=7>

Step 2: Create your custom action hook file.

2a: All custom action hook files must be located in the following directory\*:

```
/lib-hooks/__CURRENT_APP_NAME__/__ACTION_NAME__.php
```

\* The lib-hooks directory does not exist by default in our ModernBill distribution. Simply create the directory in the root location of your ModernBill installation.

2b: Replace the \_\_CURRENT\_APP\_NAME\_\_ with the name of the ORIGINAL application directory where the action file you want to override resides. (Do not use the alias name.)

Example: If you want to override or extend: /app-modernbill-client/include/lib-action/helpdesk/AddHelpdeskTicket.php, then the current application name is "app-modernbill-client".

2c: Next, name the action hook file the same as the action file name you are wanting to override or extend.

---

Example: Using the example above, the current action file is "AddHelpdeskTicket.php".

2d: You should end up with a file in the following location:

/lib-hooks/app-modernbill-client/AddHelpdeskTicket.php

Step 3: Write your action hook file using OO PHP to extend the actual action you want to override. Use the following as an example to create your action hook stub file:

## Helpdesk Integration Example

See how to use Action Hooks to integrate 3rd-party helpdesk solutions.

### Using Action Hooks for 3rd-party Helpdesk Integrations

There are several ways to integrate a 3rd-party helpdesk into the ModernBill v5 platform. In this document, we will focus on using Action Hooks and providing a high level overview of what you need to know.

ModernBill v5 has an internal helpdesk that has an Admin Interface and a Client Interface. We highly recommend starting with the Client Interface first for a basic integration, and then moving to the Admin Interface for a more advanced integration.

Note: This document assumes you have already read Writing Action Hooks.

Client Interface Action: [ShowDashboard](#)

The ShowDashboard action displays all of the Open Tickets for a client. You will want to write an Action Hook file that overwrites the addTemplate method to populate the following helpdesk variables for display in our ShowDashboard.tpl template. Our Action File: app-modernbill-client/include/lib-action/dashboard/ShowDashboard.php

Your Action Hook File: lib-hooks/app-modernbill-client/ShowDashboard.php

Example Action Hook File Contents:

Important: The parent::addTemplate method in the Action Hook will call our ShowDashboard.tpl template. You are not overwriting our entire template! You are simply overwriting the variables our template uses to display the following block of HTML:

```
{foreach key=id item=ticket from=$helpdeskTickets}
  {if $ticket.hdTicketStatusID==2}
    {$ticket.hdTicketKey}
    {$ticket.hdTicketSubject|mb_truncate:25}
    {$ticket.hdTicketDateCreated|getDateTime:short}
    {$ticket.hdTicketDateModified|getDateTime:short}
  {/if}
{/foreach}
```

Notice how the template above builds an action link to the "ShowHelpdeskTicketDetails" action which will pass in the "hdTicketID" argument. When a client clicks the ticket key link, they will be

---

redirected to the details of that ticket. In the next Action Hook, you will overwrite the execute method and write your own smarty template for display.

#### Client Interface Action: [ShowHelpdeskTicketDetails](#)

The ShowHelpdeskTicketDetails action displays a ticket's details using the hdTicketID argument. You will want to write an Action Hook file that overwrites the execute method to fetch the ticket details from your database; then prepare all smarty variables for display in your custom HookShowHelpdeskTicketDetailsAction.tpl template. Our Action File: app-modernbill-client/include/lib-action/helpdesk/ShowHelpdeskTicketDetails.php

Your Action Hook File: lib-hooks/app-modernbill-client/ShowHelpdeskTicketDetails.php

Example Action Hook File Contents:

Your Template Hook File: lib-themes/default/app-modernbill-client/templates/helpdesk/HookShowHelpdeskTicketDetails.tpl

Example Template Hook File: (Remember, this is a smarty template, not a PHP file.)

```
{* START TPL FILE *}
{include file="$tplDirectory/header.tpl" tabMajor="support" tabMinor="helpdesk"}
__WRITE_TICKET_DETAILS_HTML_HERE__
```

```
__WRITE_TICKET_UPDATE_FORM_HERE__
```

```
{include file="$tplDirectory/footer.tpl"}
{* END TPL FILE *}
```

Notice how the template above builds a form that will submit to the helpdesk.php file. It must pass AT LEAST the following arguments:

```
action = EditHelpdeskTicketResponse [Required]
hdTicketID = __HELPDESK_TICKET_ID__ [Required]
sid = {$sid} [Required]
```

The helpdesk.php file is OUR file, not yours. It will pass these arguments to the next action that you will override using Action Hooks.

#### Client Interface Action: [EditHelpdeskTicketResponse](#)

The EditHelpdeskTicketResponse action will update a ticket's status or apply additional comments entered by the client. You will want to write an Action Hook file that overwrites the execute method to perform these modifications to the ticket and update your database accordingly. Once your update is completed, you will want to redirect the client to the helpdesk ticket details action.

Our Action File: app-modernbill-client/include/lib-action/helpdesk/EditHelpdeskTicketResponse.php

Your Action Hook File: lib-hooks/app-modernbill-client/EditHelpdeskTicketResponse.php

Example Action Hook File Contents:

---

## Client Interface Action: Summary

Now that you have written three action hooks and one hook template, you can package your integration for distribution.

Your package should include AT LEAST the following 4 files\* for a complete client side integration:

```
/lib-hooks/app-modernbill-client/ShowDashboard.php  
/lib-hooks/app-modernbill-client/ShowHelpdeskTicketDetails.php  
/lib-hooks/app-modernbill-client/EditHelpdeskTicketResponse.php  
/lib-themes/default/app-modernbill-  
client/templates/helpdesk/HookShowHelpdeskTicketDetails.tpl
```

When a ModernBill user extract the files above into their installation, the client side helpdesk will interact with your application directly!