



# **Miva Merchant**

**API Product**

**Developer's Guide**

*Revision 1.1*



Miva Corporation  
2629 Ariane Drive  
San Diego, CA 92117

Telephone: 858-490-2570

Telefax: 858-490-0548

<http://www.miva.com>

[info@miva.com](mailto:info@miva.com)

This document and the software described by this document are copyright 2000 by Miva Corporation. All rights reserved. Use of the software described herein may only be done in accordance with the License Agreement provided with the software. This document may not be reproduced in full or partial form except for the purpose of using the software described herein in accordance with the License Agreement provided with the software. Information in this document is subject to change without notice. Companies, names and data used in the examples herein are fictitious unless otherwise noted.

Miva is a registered trademark of Miva Corporation. Miva Order, Miva Merchant, Miva Mia, Miva Empresa, the Miva "blades" logo, and the Miva Engine are trademarks of Miva Corporation. Windows is the registered trademark of Microsoft Corporation. All other trademarks are the property of their respective owners. This document was developed and produced in San Diego, CA, USA.

MIVA CORPORATION WILL NOT BE LIABLE FOR (A) ANY BUG, ERROR, OMISSION, DEFECT, DEFICIENCY, OR NONCONFORMITY IN MERCHANT OR THIS DOCUMENTATION; (B) IMPLIED MERCHANTABILITY OF FITNESS FOR A PARTICULAR PURPOSE; (C) IMPLIED WARRANTY RELATING TO COURSE OF DEALING, OR USAGE OF TRADE OR ANY OTHER IMPLIED WARRANTY WHATSOEVER; (D) CLAIM OF INFRINGEMENT; (E) CLAIM IN TORT, WHETHER OR NOT ARISING IN WHOLE OR PART FROM MIVA CORPORATION'S FAULT, NEGLIGENCE, STRICT LIABILITY, OR PRODUCT LIABILITY, OR (F) CLAIM FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES, OR LOSS OF DATA, REVENUE, LICENSEES GOODWILL, OR USE. IN NO CASE SHALL MIVA CORPORATION'S LIABILITY EXCEED THE PRICE THAT LICENSEE PAID FOR MERCHANT.

# Table of Contents

---

## 1 - About This Guide

Overview.....	5
Who Should Read This Guide .....	5
Guide Organization.....	5
Related Publications .....	5

## 2 - Miva Merchant™ Architecture

Overview.....	6
Miva Merchant Module Types.....	7
Commerce Libraries .....	8
Complementary Products .....	9

## 3 - Miva Merchant API Structure

Overview.....	10
Merchant2 Data Directory.....	11
Merchant2 HTML Directory.....	11
graphics .....	12
lib .....	12
modules .....	12

## 4 - Commerce Libraries

Overview.....	16
Function Variables .....	16
Commerce Initialization .....	17
Miva Commerce Loop.....	17
Describing an Error Message .....	18
The <MvCOMMERCE> Tag .....	18
An Example .....	20
Processing the Data to be Sent .....	20
Send the Message to the Server .....	21
Fields as Name-Value .....	Pairs22
Variables.....	23
Output Variables .....	23
..... Input Variables	23
Sample Files .....	24
Compiling a Commerce Library .....	25
Header File .....	25
Procedure .....	25

# Table of Contents

---

## **5 - Complementary Products**

What Are Complementary Products? .....	27
Miva Merchant Log Files .....	27
The malf.log format .....	28
Order Export File Format (orders.dat) .....	29

# Chapter 6

## About This Guide

---

### Overview

Miva Merchant is a browser-based storefront and catalog development system which uses modules to provide the look, feel and functionality of the store.

The module concept provides a true opportunity for third-party developers to create their own modules that supply additional capabilities for Miva Merchant.

### Who Should Read This Guide

This guide is for the developer who wants to build Miva Merchant modules. The guide assumes you have a general knowledge of Miva Script and are familiar with the HyperText Markup Language (HTML).

### Guide Organization

This guide is organized as follows:

Section	Description
Miva Merchant Architecture	Describes the various components of Miva Merchant and how they relate and interact with each other.
Miva Merchant API Structure	Gives the structure of the Miva Merchant API and explains the directories and files.
Commerce Libraries	Explains how libraries are used in Miva Merchant and gives an example.
Complementary Products	Describes the types of products that do not directly interface with Miva Merchant, but add features the store owners can use.

### Related Publications

For supplementary information, refer to the Miva website found at <http://www.miva.com/docs/merchant>.

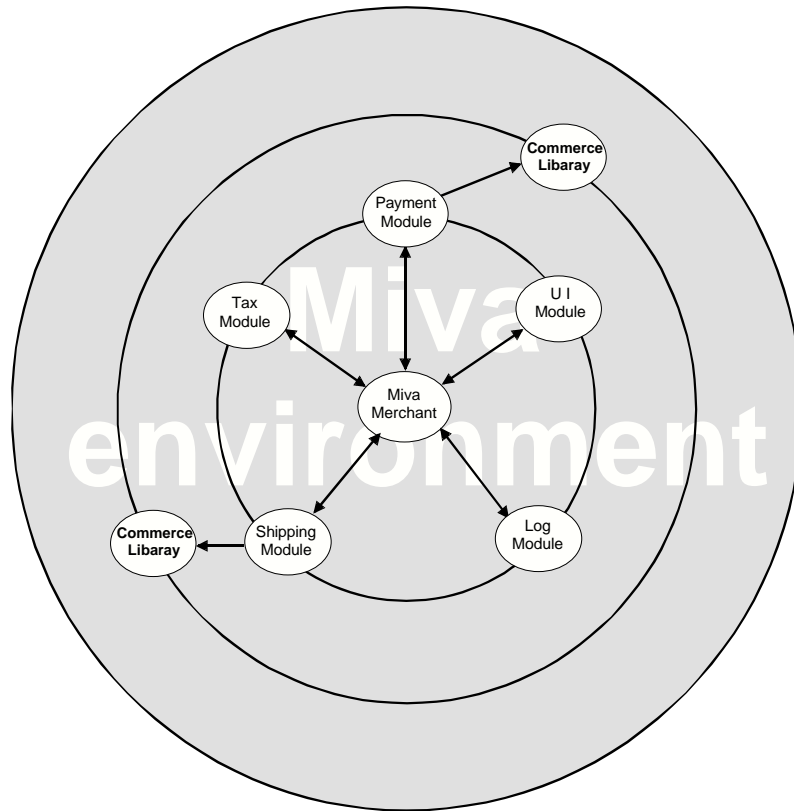
# Chapter 7

## Miva Merchant™ Architecture

---

### Overview

Miva Merchant™ is the kernel around which a series of modules provide additional functionality without any changes to the core system. Developers can build modules in XML-based Miva Script™ that provide additional features and links to outside systems such as back-end accounting functions. The figure below shows the architecture of the Miva environment.



The following table gives an explanation of the components of the figure.

Miva environment	A language preprocessor that parses and executes Miva Script applications within Miva Empresa™ on the Internet or Miva Mia™ on a PC. Miva Empresa must be running on the ISP host (or Miva Mia must be running on the PC on which Miva Script is to run) for Miva Merchant to execute.
Miva Merchant Modules	A Miva Script application that runs in the Miva environment. Provide features and functionality for Miva Merchant. The look, feel and functions of the store come from the modules that the store owner has selected.
Commerce Library	Gives Miva Merchant functionality needed when it communicates with a website that is not on the store's ISP. A commerce library is used to provide the data exchange. Commerce libraries can also be used to enhance the functionality of Miva Script.

## Miva Merchant Module Types

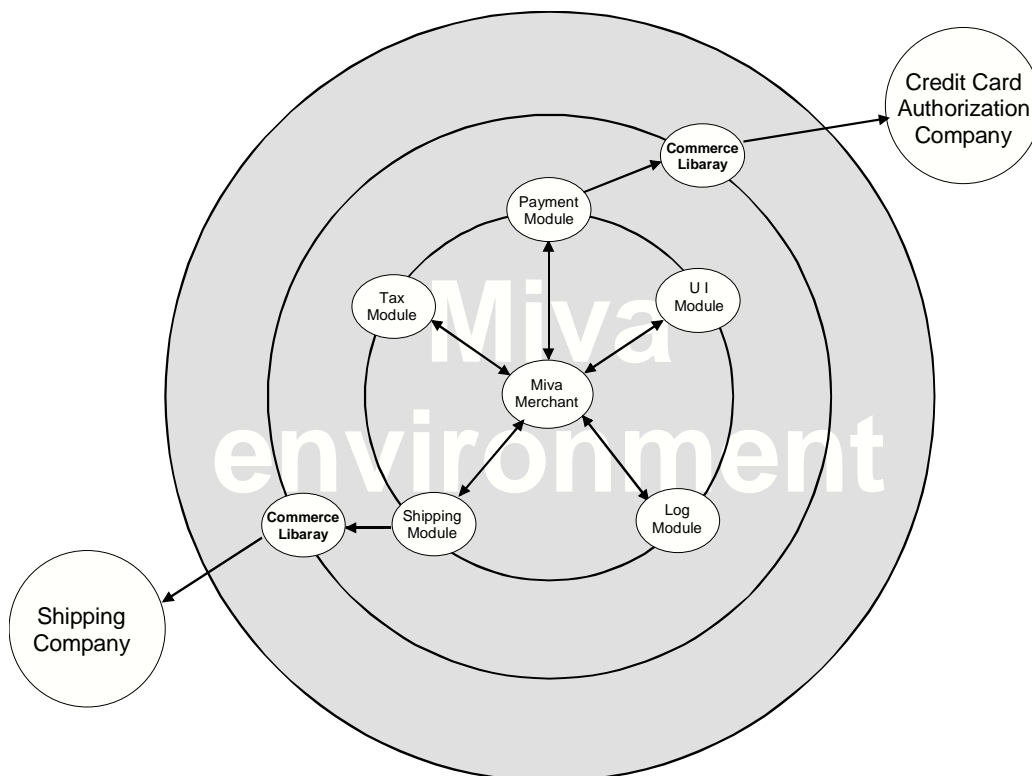
There are 13 module types that are currently available to a store owner when a store is created. The modules provide the following functions for the store.

Currency	Includes currency formatting, and cross-currency capabilities that allow store owners to do business internationally.
Data Export	Allows a store owner to export data to a PC or another site on the Internet.
Fulfillment	Provides e-mail notification of an order. This may be as simple as sending a pick slip to the warehouse, sending an e-mail to a company to drop-ship the order, or as complex as configuring a computer system and creating a bill of materials for the product.
Data Import	Allows a store owner to import data from a PC another site on the Internet.
Logging	Creates a log file. There are currently two log files available for each store.
Order Reporting	Creates a report from the order file. This type of module can create reports that use any of the data fields in the order file.
Payment Processing	Provides the necessary data to use a payment processing system, such as a credit card approval company.
Shipping	Provides the necessary information to assess the cost of shipping with a particular shipper.
Store Utility	Creates any type of utility for a store that is not covered in the other 12 types of modules.
System Extension	Provides added functionality to Miva Script.

Tax	Calculates state, county and local taxes applicable to the order.
User Interface (UI)	Provides look and feel of the store. There are currently two UIs available: KoolCat and Miva Merchant.
Utility	Provides miscellaneous utilities that do not fit into the other module types.

## Commerce Libraries

Commerce libraries are used for modules that communicate with an outside Web location for information when using protocol other than the HTTP protocol, or when adding enhanced functionality to Miva Script. The figure below shows two cases where a module would use a commerce library. The first is a payment module that processes credit card authorizations from a processing company. The second is a shipping module that would look up the cost for shipping from the shipping company's database. See Chapter 4 for more details about commerce libraries.



<MvCOMMERCE> is the Miva Script tag that begins the commerce block that enables the module to access the functionality of the commerce libraries. A commerce library is usually written in the C programming language.

When the Miva engine encounters the `<MvCOMMERCE>` tag in a script, it consults its list of registered commerce libraries. If a matching commerce library is found, Miva loads the appropriate `.dll`.

## Complementary Products

Complementary products are programs that add functionality to Miva Merchant but are not a direct part of the product. Examples are back-end accounting packages, sales log statistical programs, customer information statistical programs, creating password-protected pages, search functions and any other utilities that deliver extra performance to Miva Merchant.

# Chapter 8

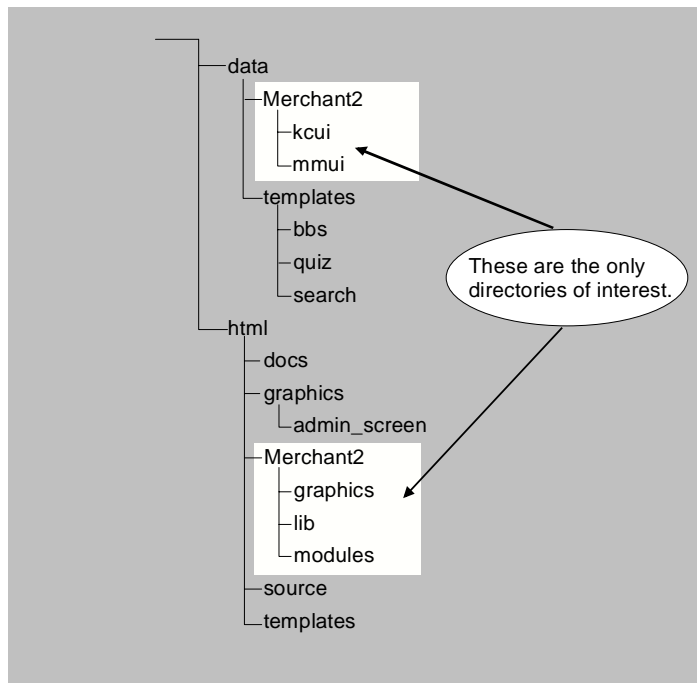
## Miva Merchant API Structure

---

### Overview

When Miva Merchant is installed, either on the Internet or on your personal computer, it builds a directory structure that contains the *Merchant2* data files, modules, and graphics files.

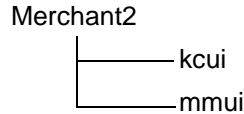
In this chapter the figures show an installation on a PC with Windows® 95/98. The figure shown below has two directory structures; *data* and *html*. These directories show the location of the *Merchant2* directories and only the two directory structures that are highlighted are relevant to the Miva Merchant API structure. Data and module files may be in directories with other names - not necessarily *data* and *html*.



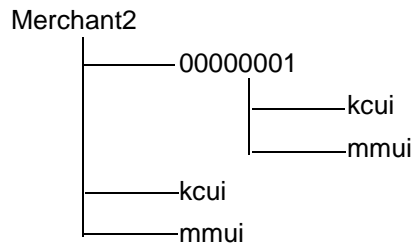
In the example above, the *data* directory contains data for a Miva Merchant store. The *html* directory contains the html scripts, the graphics for Miva Merchant and the store(s), and the Miva Merchant modules. A module is an application that can be run by either Miva Empresa or Miva Mia.

## Merchant2 Data Directory

Initially, the *Merchant2 data directory* holds the data for two user interfaces (UI). This data is stored in *kcui* for the KoolCat UI and *mmui* for the Miva Merchant UI.

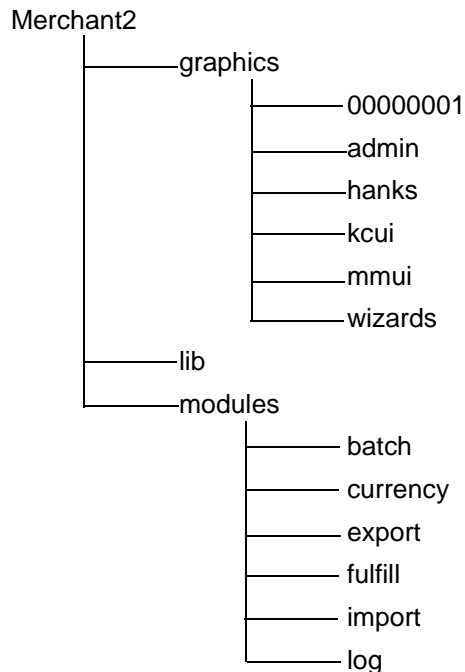


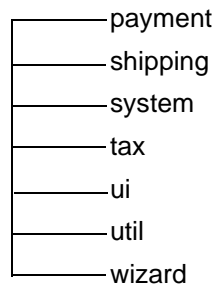
When you add a store, the system adds another directory titled 00000001. This directory holds the data that is used by the modules that you add to the store. As you add a new module to the store a new subdirectory is added to the store directory with the name of the module type. Each new store that is added gets its own directory (00000001, 00000002, etc.).



## Merchant2 HTML Directory

The *Merchant2 HTML directory* holds the Miva Merchant modules and any graphic files that are required.





*Merchant2* has three subdirectories:

- graphics
- lib
- modules

### **graphics**

Initially, the graphics directory has five subdirectories:

admin	Graphics for the administration module.
hanks	Graphics required for the demo store, Hank's Hardware.
kcui	Graphics for the KoolCat UI.
mmui	Graphics for the Miva UI.
wizards	Graphics for the Miva Merchant wizards.

When you upload a graphic to the first store, the system adds another directory, titled 00000001. This directory holds the graphics for the store. Each new store, in which a graphic is uploaded, gets its own graphics directory (00000002, 00000003, etc).

### **lib**

*lib* contains the following three Miva Active Document files:

- config.mv
- db.mv
- util.mv

### **modules**

The list below gives the directory, the file name, and the module name as shown in Miva Merchant administration. You can study the code in each module to see how you might write your module.

Directory	File	Module Name
batch	stdacct.mv	Standard Batch Report

currency	eurocur.mv	European Currency Formatting
	gencurr.mv	Generic Currency Formatting
	usmoney.mv	US Currency Formatting
export	flatord.mv	Export Orders To Flat File
fulfill	custeml.mv	Customer Order Confirmation
	meremail.mv	Email Email Merchant Notification
import	kcimport.mv	KoolCat v1.x Data Import
	mm1impt.mv	Miva Merchant v1.x Data Import
	prodimpt.mv	Import products From Flat File
log	elf.mv	e-Urchin Log
	malf.mv	Miva Merchant Access Log
payment	anacom.mv	Anacom Payment Services
	authnet.mv	Authorize Net Payment Services
	check.mv	v3.0
	cybrcash.mv	Check Payment
	ics2.mv	Cybercash Payment Services
	linkpt.mv	Cybersource ICSv2 Payment Services
	mod10.mv	Cardservice/LinkPoint Payment Gateway
	paynet.mv	Credit Card Payment With Simple Validation Signio Payflow Pro (PaymentNet)
shipping	baseunit.mv	Base + Weight Shipping
	flatrate.mv	Flat Rate Shipping
	minunit.mv	Minimum or Weight Shipping
	qship.mv	Quantity-based Shipping
	ups.mv	UPS OnLine Shipping Calculation
system	example.mv	There are no system modules that come with Miva Merchant.
tax	shoptax.mv	Shopper Selected State Tax
	statetax.mv	State Based Sales Tax
	vat.mv	VAT (Value Added Tax)
ui	kcui.mv	KoolCat v1.x Look and Feel
	mmui.mv	Miva Merchant Look and Feel -
util		There are no utility modules that come with Miva Merchant.

## wizard

atwizard.mv	Add Product Attribute Wizard
cawizard.mv	Add Category Wizard
cswizard.mv	Create Store Wizard
pawizard.mv	Add Product Wizard
pcwizard.mv	Payment Configuration Wizard
shwizard.mv	Shipping Configuration Wizard
uiwizard.mv	Look & Feel Configuration Wizard

These modules can be seen on the Miva Merchant administration page if you click on Modules.

**To learn how modules work, use the following procedure.**

1. Add a module to a store.
2. Review the look, feel and functionality of the module.
3. From the directory structure shown previously, find the module name.
4. Open the module using a code editor.
5. Review the code to see how the module works.



# Chapter 9

## Commerce Libraries

---

### Overview

A Miva commerce library is the same as other C libraries. It must contain all of the functions required by the program that will process the data that is exported from the commerce library. A Miva commerce library is called by the `<MvCOMMERCE>` tag in a module.

### Function Variables

A commerce library must include the following four functions:

- `miva_commerce_init`
- `miva_commerce_loop`
- `miva_commerce_cleanup`
- `miva_commerce_error`

Each of the functions requires a set of data variables that contains the data. The following table identifies these variables.

---

**Note:** Not all functions contain every variable.

---

<code>context</code>	points to the handle of the file that contains the operating state information for Miva Empresa. A commerce library needs this handle to access the networking and file functions provided by the commerce API.
<code>data</code>	a placeholder that was passed to <code>miva_commerce_init</code> . Value assigned in <code>miva_commerce_init</code> , will be the same here.
<code>method</code>	the value of the <code>METHOD</code> attribute that was specified in the <code>&lt;MvCOMMERCE&gt;</code> tag.
<code>action</code>	the value of the <code>ACTION</code> attribute that was specified in the <code>&lt;MvCOMMERCE&gt;</code> tag.
<code>input</code>	a handle to <code>Miva_VariableList</code> containing all the variables specified in the <code>FIELDS</code> parameter of the <code>&lt;MvCOMMERCE&gt;</code> tag. (This makes up the majority of the input to a commerce tag).
<code>output</code>	a handle to <code>Miva_VariableTree</code> which is used by the commerce library to return values to the script. All variables added to this tree are made available to the script in the form of system variables, which are available only inside the <code>&lt;MvCOMMERCE&gt;</code> block.
<code>iteration</code>	the number of times Miva has executed the code inside the <code>&lt;MvCOMMERCE&gt;</code> block. The first time <code>miva_commerce_loop</code> is called, the iteration has a value of 0.

## Commerce Initialization

The `miva_commerce_init()` function is called before entry into an `<MvCOMMERCE>` block. It is used to initialize any information or connections used by `miva_commerce_loop`.

For “An Example” on page 20 (`example.c`), all of the processing is done in `miva_commerce_init`.

```
Miva_Commerce_Status miva_commerce_init(
Miva_Context context,
void**data,
const char *method,
const char *action,
Miva_VariableList input,
Miva_VariableTree output )
```

Assuming `miva_commerce_init` returns `MIVA_COMMERCE_OK` each time it reaches the end of the `<MvCOMMERCE>` block (the `</MvCOMMERCE>` tag), Miva Empresa calls the `miva_commerce_loop` function.

## Miva Commerce Loop

The `miva_commerce_loop()` function is required in the commerce library only when a loop is necessary. It is called before each iteration of the `<MvCOMMERCE>` tag (including the first entry into the tag). The iteration parameter indicates the number of times the function has looped. It is 0 for the first iteration.

A return of `MIVA_COMMERCE_OK` continues looping, or `MIVA_COMMERCE_END` stops the looping.

```
Miva_Commerce_Status miva_commerce_loop(
Miva_Context context,
void **data,
const char *method,
const char *action,
Miva_VariableList input,
Miva_VariableTree output,
int iteration )
```

The action that Miva Empresa takes is dependent on the return value your coding sends back to `miva_commerce_loop`.

Return Value	Action
<code>MIVA_COMMERCE_OK</code>	Miva Empresa executes the code inside the <code>&lt;MvCOMMERCE&gt;</code> block.
<code>MIVA_COMMERCE_ERROR</code>	Miva Empresa exits the block and sets the <code>MvCOMMERCE_Error</code> variable (it can display the message and/or terminate the entire script, depending on the settings).
<code>MIVA_COMMERCE_END</code>	Miva Empresa will exit the commerce block normally ( <code>MIVA_COMMERCE_END</code> normally returned the commerce library completed its purpose).

When the commerce library finishes, regardless of the return value of `miva_commerce_init` or `miva_commerce_loop`, it calls `miva_commerce_cleanup`.

## Cleaning Up Allocated Resources

The `miva_commerce_cleanup()` function is responsible for cleaning up any resources allocated by this library. Note that this function is called even if `miva_commerce_init` or `miva_commerce_loop` returns `MIVA_COMMERCE_ERROR`.

```
void miva_commerce_cleanup(
Miva_Context context,
    void **data,
    const char *method,
    const char *action,
Miva_VariableList input )
```

Return Value	Action
<code>MIVA_COMMERCE_ERROR</code>	Miva Empresa calls <code>miva_commerce_error</code> to receive a description of the error. It makes this call prior to calling <code>miva_commerce_cleanup</code> .

## Describing an Error Message

The `miva_commerce_error()` function returns a pointer to a string describing the last error message that occurred.

```
const char *miva_commerce_error(
Miva_Context context,
    void **data,
    const char *method,
    const char *action,
Miva_VariableList input )
```

## The <MvCOMMERCE> Tag

The `<MvCOMMERCE>` tag provides the interface to an outside Web location (commerce server). As mentioned earlier, those modules that require communication to a commerce server do so with the use of a commerce library. Values are passed between the module script and commerce library in the form of variables.

Depending on which services you have requested, `<MvCOMMERCE>` loops one or more times.

Each service has its own detailed requirements, but you must request all services using the same basic approach. The basic format of `<MvCOMMERCE>` looks like:

```

<MvCOMMERCE
  ACTION="url_to_commerce_server"
  METHOD  ="access_method"
  FIELDS="value1,value2,...">
...
...<MvCOMMERCESTOP>...
...
</MvCOMMERCE>

```

where:

<b>ACTION</b>	the URL for the destination server providing services. This is not required. It can be handled in the commerce library code.
<b>METHOD</b>	identifies the type of service you are requesting; for example, <code>UPSCost</code> (UPS Quick Cost).
<b>FIELDS</b>	contains the list of fields that make up the message that you are sending for processing. Information about the services being requested, the merchant requesting the services, and the customer is sent in the message as a list of variables.
<b>&lt;MvCOMMERCESTOP&gt;</b>	(optional) explicitly terminates the execution of <code>&lt;MvCOMMERCE&gt;</code> . This is an empty tag.

The parameter input to `miva_commerce_init` is a dynamically linked list of variables that are passed as the `FIELDS` attribute of the `<MvCOMMERCE>` tag.

The parameter output from the `miva_commerce_init` and `miva_commerce_loop` functions is a binary tree that allows a commerce library to make results available to a commerce module.

You may embed Miva Script code directly into an HTML document that contains form code, or you may use Miva Script in a separate processing application. Either way, you must have a data input form that collects data from your end users using a URL to an application that is processed by Miva Empresa as its `FORM ACTION` value.

To set up commerce processing, use the following process:

1. Create a form that contains fields for each input variable (that is, each type of information) that you want to send to export.
  - a. The name (`NAME` attribute) of each field must be the same as the corresponding variable name.
  - b. Choose form objects that are appropriate to the type of data. For example, check boxes for yes/no choices, text boxes for text, and radio buttons or drop-down lists for choices from a defined group.
  - c. Set the form's `ACTION` attribute to point to the Miva Script program containing the `<MvCOMMERCE>` tag. If it's the same as the script containing the form, use the macro `&[documenturl]`; as the value of `ACTION`.

Set the form's `METHOD` attribute to `POST`.

- d. Include each of the input variables from the form in the `FIELDS` attribute of `<MvCOMMERCE>`. You only need to include those variables that actually appear in the form.
2. Between the `<MvCOMMERCE>` and `</MvCOMMERCE>` tags, insert code that processes the information sent back in the output variables. You don't have to use all of these variables in your code, just the ones you need. The processing that you do can be as simple as just displaying values of certain variables, or you can do more complex processing, as desired. At a minimum, you should display the returned data, or an error message (`errmsg`), if there is one.

When an HTML document containing Miva Script code is in a place where Miva Empresa can process it, that document becomes a web application. This web application has the ability to process the data and perform operations on the data — not just display or collect data.

## An Example

The following code is a concept example. Some actual code is omitted (...code...) for the purpose of expediency in showing the code's structure.

### Processing the Data to be Sent

```
<HTML>
<HEAD><TITLE>Order Form</TITLE></HEAD>
<BODY BGCOLOR = "#ffffff">

<MvCOMMENT>
Test to see if this is the first time the program is run. If the
value of form_done is greater than zero, data collected with the
form during the previous iteration is processed. If the value of
form_done is not greater than zero, Miva drops through to the form
to collect data.
</MvCOMMENT>

<MvIF EXPR="{ form_done GT 0 }">
  <MvCOMMERCE ACTION="http://...URL_to_Commerce_Server..."
    METHOD="Commerce_Protocol"
    FIELDS="
      comma,
      delimited,
      list,
      of fields..."
  >
  ...More Miva Script Code to test for return values sent back from
  the commerce Server, display results out to the browser, process
  data with internal systems, etc...
  </MvCOMMERCE>
  <MvEXIT>
</MvIF>

<MvCOMMENT>
```

This is the first display generated in the browser if it is the first time that the program is run.

```

</MvCOMMENT>
<H1>Order Form</H1>
<FORM ACTION="http://...URL_to_this_file..." METHOD="post">
  <B>First Name:</B><BR>
  <INPUT TYPE="text" NAME="fname" SIZE = 30>
  <P>
  <B>Last Name:</B><BR>
  <INPUT TYPE="text" NAME="lname" SIZE = 30>
  <P>
<MvCOMMENT>
This is the HTML code for other form fields like product
selections, etc.
</MvCOMMENT>
  <P>
    <B>Credit Card Number:</B><BR>
    <INPUT TYPE="text" NAME="ceditcard" SIZE = 30>
    <P>
      <B>Expiration Date: Name:</B><BR>
      <INPUT TYPE="text" NAME="expiredate" SIZE = 5>
      <P>
        <B>Card Holders Name:</B><BR>
        <INPUT TYPE="text" NAME="cardholdrname" SIZE = 30>
        <P>
          <INPUT TYPE = "SUBMIT" VALUE = "Fill My Order">
          <MvASSIGN NAME="form_done" VALUE="{ 1 }">
          <MvHIDE FIELDS="form_done">
</FORM>
</BODY>
</HTML>

```

In the code shown above, the HTML form code and the Miva Script code that process the form data are placed together in the same document. This is the recommended method of implementing HTML forms and Miva Script code. More advanced implementations may test for NULL values or improper entries in certain fields. Based on such conditional processing, you may want to redisplay the form to the user for re-entry before processing the data with the vendor server.

### Send the Message to the Server

The server application that receives the variable list checks the list against the list of services authorized for the requesting merchant. If the list is valid, the application performs the services and returns the results for all services in a single message.

Check the fields in the returned message for status and data.

```

<MvCOMMERCE ACTION="http://...URL_to_CSP_Server..."
  METHOD="access_method"
  FIELDS="...list_of_fields">
  ...code...

```

```

        ...code...
        ...code...
<MvIF EXPR = "{ rcode LT 0 }">
    Order Number
    <MvEVAL EXPR = "{ order_number }">
        failed because
    <MvEVAL EXPR = "{ err }">
</MvIF>
    <MvIF EXPR = "{ rcode EQ 0 }">
        Order Number
        <MvEVAL EXPR = "{ order_number }">
            was declined because
        <MvEVAL EXPR = "{ declined }">
</MvIF>
    <B>Your Order was Processed</B><BR>
    <MvIF EXPR = "{ download_url0 }">
        <B>Download URL:</B><BR>
        <A HREF = "&[download_url0]">
        <MvEVAL EXPR = "{ download_url0 }"></A>
    <P>
</MvIF>
<MvIF EXPR = "{ order_number }">
    Order Number:
    <MvEVAL EXPR = "{ order_number }"><BR>
</MvIF>
<MvIF EXPR = "{ preapp_auth_code }">
    Authorization Code:
    <MvEVAL EXPR = "{ preapp_auth_code }"><BR>
</MvIF>
<MvIF EXPR = "{ preapp_auth_time }">
    Authorization Time:
    <MvEVAL EXPR = "{ preapp_auth_time }"><BR>
</MvIF>

<MvIF EXPR = "{ preapp_merch_txn }">
    Merchant Transaction #:
    <MvEVAL EXPR = "{ preapp_merch_txn }"><BR>
</MvIF>

```

## Fields as Name-Value Pairs

The server responds with a return message containing a different set of name-value pairs which provides information on the results of performing the services requested in the original message.

When sending messages via Miva modules, all the work of building name-value pairs is done automatically through Miva Empresa's Preprocessor. HTML form variables automatically become Miva variables when submitted to a Miva active document through the use of the

```
<FORM ACTION="http://..." METHOD="(POST or GET)">
```

construct. By properly using the `<MvCOMMERCE>` tag in a Miva active document, the values passed to the active document are already formatted into name-value pairs as they are processed by Miva — and subsequently passed to the server for processing.

## Variables

This section gives examples of output variables that are sent to the processing application and input variables that receive the data from the processing application. In the examples below, a shipping type module is used. Therefore, the variables shown are only within a shipping module and are presented here to show you how variable may be used.

### Output Variables

As an example, the following output variables are available inside the `<MvCOMMERCE>` block of a shipping module:

Name	Value
errmsg	Description of error, or NULL if no error occurred
errorcode	Numeric code of error, or NULL if no error occurred
message	Informational message
product	Product code
orig_postal	Shipment source postal code
orig_country	Shipment source country code
dest_postal	Shipment destination postal code
dest_country	Shipment destination country code
zone	Shipping zone
weight	Shipment weight
productchrg	Shipment charge, minus any accessory or surcharges
accs_surchar g	Shipment accessory or surcharges
totalchrg	Total cost of shipment
time	Commit time, or -1 if end of day

### Input Variables

The following list gives the input variables that may be in a shipping form and passed to the processing calculator using the `FIELDS` attribute. The Value column describes the types of values that each variable must be given in the form.

---

**Caution:** Any variable that starts with a number must be prefixed with `g.` (global) or `l.` (local) if they are used in a Miva Script expression. For example, `g.25_length + g.26_width`.

---

Name	Value
accept_license_agreement	Must have the value 'yes' when the form is submitted.
13_product	Product Code corresponding to the type of shipment service desired. For example, '1DA' for Next Day Air.
10_action	Set to '3' to obtain information only for the service specified with 13_product, or '4' to obtain information on all applicable services.
15_origPostal	The ZIP code of the originating location (U.S. only).
19_destPostal	The ZIP or other postal code of the destination location (all countries, as applicable).
22_destCountry	The two-letter country code of the destination country: for example, 'US' for the United States, 'CA' for Canada, 'MX' for Mexico.
23_weight	Weight in pounds.
24_value	Value in dollars.
25_length	Length in inches.
26_width	Width in inches.
27_height	Height in inches.
29_oversized	'1' if the package is oversized, '0' if it is not.
30_cod	'1' if the shipment is COD, '0' if it is not.
33_hazard	'1' if the package contains a hazardous material, '0' if it does not.
34_handling	'1' if the package requires additional handling, '0' if it does not.
35_calltag	'1' for basic call tag service, '2' for electronic, '0' for none.
37_saturdaydelivery	'1' for Saturday delivery, '0' otherwise.
38_saturdaypickup	'1' for Saturday pickup, '0' otherwise
39_response	'1', '2', '3', '4' for various proof of delivery responses, '0' for none.
43_vcd	'1' for verbal confirmation of delivery, '0' otherwise.
44_FirstShipNotify	First Ship Notification Service: '1' for Domestic, '2' for International, '0' for none.
45_SecondShipNotify	Second Ship Notification. '1' for Domestic, '2' for International, '0' for none.

## Sample Files

The following files are examples of various files associated with Miva modules and commerce libraries. *example.mv* and *ups.mv* show the use of the <MvCOMMERCE> tag to call the commerce library.

File	Description
example.c	Example file written in C that shows examples of the code for the four function variables: miva_commerce_init miva_commerce_loop miva_commerce_cleanup miva_commerce_error
Example.def	(Windows® only) Module Reference File.
Example.dsp	Microsoft Developer Studio® Project File for example.
example.mv	Example of a payment module
Example.rc	Microsoft Developer Studio generated resource script for example.mv.
Makefile	An example Makefile for Unix.
mivapi.h	Header file.
MivAPI.lib	Library needed by Windows.
resource.h	Microsoft Developer Studio generated include file. There are two resource files, one for example.rc and one for UPSOnline.rc.
upsonline.c	Source code for upsonline, an example of a UPS® shipper package.
UPSONline.def	Module Reference File for UPS Online.
UPSONline.dsp	Microsoft Developer Studio Project File - Name for UPSOnline.
ups.mv	Example UPS module.
UPSONline.rc	Microsoft Developer Studio generated resource script for UPS Online.

## Compiling a Commerce Library

### Header File

A commerce library must include the Miva header file:

```
#include <mivapi.h>
```

### Procedure

Under UNIX, you need to compile your commerce library into a Dynamic Shared Object (DSO). This is dependent on the platform and compiler. For example, the GNU C Compiler (GCC) would use the following command line:

```
gcc -shared -o example.so example.c
```

Under Windows, you need to compile your commerce library into a Dynamic Link Library ( .dll ). Most Windows compilers have options to produce a .dll . After successful completion, link your .dll with `MivAPI.lib`.

# Chapter 10

## Complementary Products

---

### What Are Complementary Products?

Complementary products can be developed by users, system integrators and consultants to enhance Miva merchant. Some product examples include:

- Back-end accounting system
- Miva Merchant access log data statistical information
- Sales log data statistical information
- Script editors

### Miva Merchant Log Files

Miva Merchant provides two files that contain useful information for the store owner.

- Miva Merchant Access Log file
- Order Export file

The Miva Merchant access log provides the store owner with valuable statistical information for reporting and analysis. The Miva Merchant order export log can be used to export order and financial data to back-end accounting programs and order tracking programs. For an idea of what type of complementary products might be developed, we offer a description of these two file.

#### Miva Merchant Access Log File (`malf.log`)

Miva Merchant comes with an access log file. Complementary products may utilize this log file to offer Miva Merchant users reporting capabilities to enhance their store's marketing position. Since complementary products are designed to work outside of Miva Merchant, they need not be designed in Miva Script.

In the case of log files, complementary products need to be designed in a manner that can integrate Miva Merchant's log file format. The use of Miva Merchant's export functions may also provide data for complementary products, such as inventory control or tax analysis applications.

The purpose of the `malf.log` file is to record customer visits to a Miva Merchant storefront Web site. Based on the National Center for Supercomputing Applications (NCSA) Extended Combined Format, the `malf.log` file provides a nearly identical format with the addition of information into the request field.

The file records one log entry per line with each field separated by a single white space. Blank fields should be represented by the "-" character.

Except for the `referer`, and `user-agent` fields which are surrounded by quotes, fields containing white space must be surrounded by [ ] square brackets.

Fields that are identical to the NCSA format are not discussed in detail. For more information about these fields, refer to the [www.apache.org](http://www.apache.org) Web site.

The main alterations to the NCSA format come in the second, third, and fifth fields.

Field	Description
2	Changed from the remote logname to the name of the storefront. This allows one log file to contain multiple store fronts for one Web site.
3	Changed from the remote user to the session ID. The session ID uniquely identifies each customer.
5	Changed from the first line of the request header, now contains three specific sub-fields that provide additional information about the actions of the customer. If any of the sub-fields contains white space, then the sub-field must be surrounded by [ ] square brackets.

### The *malf.log* format

The modified NCSA format used by Miva Merchant in its *malf.log* file is shown below.

```
%h %{STORE} %{SESSIONID} %t \"%{ACTION} %{SCREEN} %{ATTRIBUTE S}"
%s %b \"%{Referer}i\ " \"%{User-agent}i\ "
```

where:

%h	the remote host (see <a href="http://apache.org">apache.org</a> for additional information).
%{STORE}	the name/ID of the storefront.
%{SESSIONID}	the unique session identifier of the customer.
%t	time in the common log format (see <a href="http://apache.org">apache.org</a> for additional information).
%{ACTION}	an identifier describing the action the user has taken.
%{SCREEN}	an identifier describing the screen the user is shown.
%{ATTRIBUTE S}	a description of the screen attributes possibly including product names.
%s	the status code (see <a href="http://apache.org">apache.org</a> for additional information).
%b	the bytes sent (see <a href="http://apache.org">apache.org</a> for additional information).
%{Referer}	the referring URL (see <a href="http://apache.org">apache.org</a> for additional information).
%{User-agent}	the user-agent (see <a href="http://apache.org">apache.org</a> for additional information).

Below is an example of a single record as recorded by the *malf.log* file:

```
remotehost.net "001" 01BF41C24B108D20FFF73BA90000000 [08/Dec/
1999:13:22:16 -0800] - SFNT "001" 200 - "http://www.somesite.com/
Merchant2/merchant.mv" "Mozilla/4.0 (compatible; MSIE 4.01; Windows
98)"
```

## Order Export File Format (`orders.dat`)

The order file records purchase and shipping/billing information of all on-line transactions to the store. For security reasons, the `orders.dat` file does not record credit card information.

Complementary products may use this data to provide basic accounting services, tax accounting, shipping analysis, purchased product analysis and reporting, and customer lists.

The format is in a tab-delimited, single-line format. The `orders.dat` file records the following fields:

ORDER_ID	the order ID.
PROCESSED	the processed code (1 = processed, 0 = unprocessed).
ORDER_DATE	the date of the order.
ORDER_TIME	the time of the order.
SHIP_FNAME	the first name of the customer in the Shipping data.
SHIP_LNAME	the last name of the customer in the Shipping data.
SHIP_EMAIL	the e-mail address of the customer in the Shipping data.
SHIP_COMP	the company name of the customer in the Shipping data.
SHIP_PHONE	the phone number of the customer in the Shipping data.
SHIP_FAX	the fax number of the customer in the Shipping data.
SHIP_ADDR	the shipping address of the customer.
SHIP_CITY	the Ship To City of the customer.
SHIP_STATE	the Ship To State of the customer.
SHIP_ZIP	the Ship To ZIP code of the customer.
SHIP_CNTRY	the Ship To country of the customer.
BILL_FNAME	the first name of the customer in the Billing data.
BILL_LNAME	the last name of the customer in the Billing data.
BILL_EMAIL	the e-mail address of the customer in the Billing data.
BILL_COMP	the company name of the customer in the Billing data.
BILL_PHONE	the phone number of the customer in the Billing data.
BILL_FAX	the fax number of the customer in the Billing data.
BILL_ADDR	the billing address of the customer.
BILL_CITY	the Bill To City of the customer.
BILL_STATE	the Bill To State of the customer.
BILL_ZIP	the Bill To ZIP code of the customer.
BILL_CNTRY	the Bill To country of the customer.
PROD_CODE	the Product Code.
PROD_NAME	the Product Name.
PROD_UPSLD	the Product Upsold code (1 = Upsale product purchased, 0 = Upsale Product not purchased).
PROD_PRICE	the Product Price.

PROD_QUANT	the Product Quantity.
PROD_ATTR	the Product Attribute.
PROD_OPT	the Product Option.
OPT_PRICE	the Option Price.
ORDER_TAX	the total tax amount for the order.
ORDER_SHIP	the total shipping amount for the order.
ORDER_TOTL	the total amount for the order.