# Teamwork Retail eCommerce

## Magento Integration Guidelines

DRAFT

# Table of Contents

# 1. Introduction

## 1.1 About This Guide

This guide provides information about installing and configuring Teamwork Retail eCommerce plugins to work in your environment. This guide also provides a technical description of the Teamwork Retail plugins and a list of all staging tables in use and under development.

### 1.1.1 Intended Audience

This guide is intended for developers integrating Teamwork Retail with Magento ecommerce websites.

### 1.1.2 Revision History

| Revision | Date | Contributors | Comment |
|---|---|---|---|
| 00.00.04 | 23.04.2014 | Mary K Swanson | Edited, reformatted |
| 00.00.03 | 18.04.2014 | Eugene Yasyuchenko | Restructured paragraphs |
| 00.00.02 | 26.04.2013 | Eugene Yasyuchenko | Updated document layout |
| 00.00.01 | 26.10.2013 | Vitalii Plodistov; Vitalii Chumak | Initial draft |

### 1.1.3 Abbreviations

| Abbreviation | Description |
|---|---|
| CHQ | Cloud HQ |
| EC | ecommerce |
| ECM | eCommerce Catalog Memo |
| HQ | Headquarters, the company's central operations |
| TS | Teamwork Service |
| TT | Teamwork Transfer |
| TWS | Teamwork Staging package |
| TWT | Teamwork Transfer package |
| WST | Web staging tables |

### 1.1.4 Authors

Alexander Panasovsky ap@cloudwk.com

Vitalii Plodistov vp@cloudwk.com

Eugene Yasyuchenko ey@cloudwk.com

## 1.2 About Teamwork Retail eCommerce

Teamwork Retail eCommerce is an integrated system based on the Magento ecommerce platform. It gives your company the ability to offer true omni-channel shopping by creating one experience for your customer, in-store and online.

### 1.2.1 System Overview

The Teamwork Retail eCommerce integration with Magento enables you to share a retail catalog between physical and online stores. The integration package contains the following, implemented as Magento modules:

- Teamwork_Service
- Teamwork_Transfer

These modules operate external requests to update product information in the Magento database and to provide sales information to the Magento database.

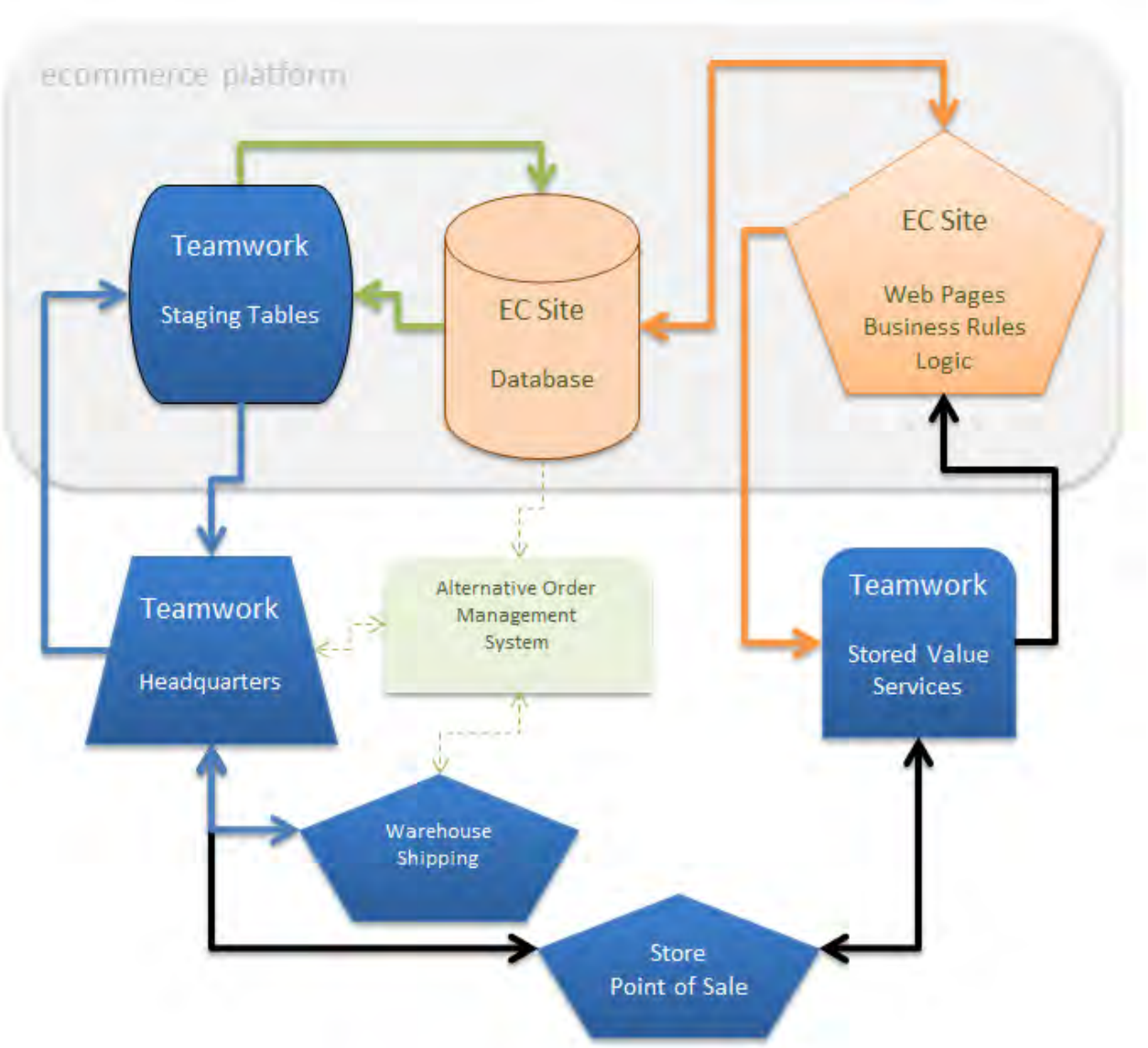
The modules create and use tables, called staging tables in Teamwork Retail eCommerce, to keep raw and transformed data for exchange in Magento. (See figure 1, Integration Workflow Overview.) There are three distinct phases of exchange:

- Staging Integration
  The interchange of information between Teamwork Headquarters and Teamwork Staging Tables. (See blue lines in figure 1.)
- Service Integration
  Serves stored value information when requested by the ecommerce site, including customer membership validation, gift card authorization, real time quantity information, and loyalty reward points tracking. (See black lines in figure 1.)
- Site Integration
  Information exchange between Teamwork Staging Tables and the ecommerce site database. (See green lines in figure 1.)

The Teamwork Retail eCommerce integration also uses special web staging tables.

**Note**

Unless otherwise named in Magento, all staging tables contain the prefix ***'service_'.***




1. Integration Workflow Overview

### 1.2.2 System Requirements

The system requirements are the following:

- MySQL 5.x or higher
- Magento 1.4.x or higher
- curl (command line tool)

Compatibility:

- Magento CE: 1.10, 1.11, 1.12, 1.13

## 1.3 Setup

### 1.3.1 Plugin Installation

For integration with Magento, your Teamwork Retail representative will provide two files based on the CHQ (Cloud HQ) version:

- TeamworkService-XX.XX.XX.tgz – Contains service plugins.

  **Note**

  Changes to the TeamworkService module are not recommended.

- TeamworkTransfer-XX.XX.XX.tgz – Contains code examples that can be changed if needed.

To install or update a Magento software package:

1. Log in using the Magento Admin panel and disable Compilation Mode, if enabled. (*System* > *Tools* > *Compilation*)
2. Log out from the Magento Admin panel.
3. Unpack the contents of the Teamwork Retail eCommerce package file to your Magento root folder.
4. Log in again to the Magento Admin panel.
5. Go to *System* > *Cache Management*.
   a. Choose *Select All* to select all types of cache.
   b. Select *Refresh* from the drop-down menu.
   c. Click *Submit*.
   d. After the page reloads successfully, click *Flush Magento Cache* and then *Flush Cache Storage*.
6. Enable Compilation Mode again, if desired.

#### 1.3.1.1 Verify Successful Installation

After installation, verify that the web staging tables and the SOAP/XML-RPC user “testservice” have been created.

### 1.3.2 Configuration

After installation, you’ll need to configure Teamwork Retail CHQ and establish a successful connection between CHQ and your ecommerce website.

1. Log into Magento Admin, go to *System* > *Manage Stores*, and set *Store View Name* to default.

   **Note**

   The CHQ channel must have the same name as *Store View Name* in Magento. The default store is the store in which id = 0.

2. Inside Teamwork Retail CHQ, go to *Options >Channels >Edit* and select or enter these field options:
    a. Upload to — Magento
    b. URI — http://{address of your Magento}/api/xmlrpc
    c. User — testservice
    d. Password — testservice
    e. API key — testservice

**Note**

Optionally, you can set the user and password in the path: *System > Web Services > SOAP/XML-RPC – Users.*

3. Send an ECM (eCommerce Catalog Memo) to Magento to get required setting.
4. If needed, map settings and payment methods inside CHQ by going to *Options >Channels >Edit*.

**Note**

Please see Teamwork Retail CHQ user guide for detailed explanation of settings.

5. Verify configuration.
    a. Create and send an ECM in CHQ. Check if updates come to the website.
    b. Create an order in Magento, and check if it comes to CHQ.

# 2. Teamwork Service Module

Teamwork Service (TWS) is a Magento extension that provides storage and communication between CHQ and a Magento website. It includes a set of staging tables where CHQ pushes the catalog and other basic information and retrieves web orders. TWS also includes synchronization services that CHQ utilizes for data exchange with the website.

## 2.1 Purpose

After installation, TWS adds staging tables to the Magento database. The staging tables contain a "service" prefix (though Magento may also add a prefix). See more info about staging tables in the Staging Tables Overview.

TWS uses staging tables to save request data and prepare responses. The module also initiates Teamwork Transfer work, which is responsible for data synchronization between staging and standard Magento tables.

## 2.2 Data Exchange Protocol

These modules use the XML-RPC (remote procedure calling) protocol. The XML-RPC specification allows procedure calls over the Internet to between different software and platforms. The XML-RPC message is an HTTP-POST request; the body of the request is in XML. When a procedure executes on the server, the value it returns is also formatted in XML.

## 2.3 Features

All RPC requests can be divided into ECM and non-ECM, or background, requests. ECM requests are designed to synchronize information such as products and categories data, discounts, and stock data. ECM requests are performed manually by an administrator or periodically using a tunable timer. Background requests are designed to retrieve operative information and usually are performed automatically. Examples include getting ECM request status or retrieving new orders.

The parameters of an RPC request can be a semicolon-delimited string or Base64-encoded XML representing a set of values. Most request parameters require the field *channel_id*, which represents the Magento store entity. If the parameter is a string, the *channel_id* field should be first. If the parameter is XML, the location of the *channel_id* field depends on the request.

See more about relationships between the modules and the Magento entities in the Teamwork Transfer module section.

## 2.4 Main Class Descriptions

Main classes in TWS are:

- Teamwork_Service_Model_Api: initial for XML-RPC request; most methods are module's entry points (see etc/api.xml).
- Teamwork_Service_Model_Service: designed to operate ECM request.
- Teamwork_Service_Model_Ecm: designed to return ECM request status.
- Teamwork_Service_Model_Status: designed to set order status (fills "service_status" and "service_status_items" staging tables and calls Teamwork Transfer module to change order status).
- Teamwork_Service_Model_Weborder: designed to return weborder (all "fresh" data from "service_weborder" staging table filled by the Teamwork Transfer module).
- Teamwork_Service_Model_Settings: designed to exchange settings data.
- Teamwork_Service_Model_Mapping: designed to exchange mapping data.
- Teamwork_Service_Model_Version: designed to return module version.
- Teamwork_Service_Helper_Data: helper class which contains methods to call Teamwork Transfer module.

## 2.5 Interaction with Teamwork Transfer Module

The TWS module calls the Teamwork Transfer module for these actions:

- to process an ECM request
- to set order status

The TWS Module uses curl for these actions. The curl call originates from the Teamwork_Service_Helper_Data helper. It can be synchronous (waiting while Teamwork Transfer module finishes work) or asynchronous (disconnecting 15 seconds after calling Teamwork Transfer controller, regardless of whether it has finished work). The URLs curl uses for both cases contain paths to the Transfer Index controller. For both cases, etc/config.xml provides this information, the *<teamwork_service><staging_url>* and *<teamwork_service><status_url>* values. Also curl adds the *request id* parameter for a synchronous call or the *package id* parameter for an asynchronous call, which helps the Teamwork Transfer module get the necessary data from the staging tables to process requests and prepare responses, if needed in the staging tables.

# 3. Teamwork Transfer Module

The Teamwork Transfer module (TWT) is an extension that moves data between the staging tables and Magento tables.

## 3.1 Purpose

The TWT module is designed to synchronize data between staging and standard Magento tables. It uses standard Magento classes while reading, creating, and updating Magento tables, which ensures additional standard and custom code execution, for example, initiating standard and custom event observers, re-indexing processes while saving products, or using a custom model/helper rewriting mechanism.

## 3.2 Features

Features of the Teamwork Transfer module:

- Most data from the staging tables that the module uses bind to the *channel id*. In turn, this parameter is linked to the Magento store. Known channels are stored in the *service_channel* table. Channels are linked to stores by *channel_name* and *store code.* (See the *getStoreByChannelId* method, a part of the Teamwork_Transfer_Model_Transfer class.)
- The TWT module always uses Magento's "default" store to set attribute values.

  **Note**

  Magento's default store is the store in which id = 0.

- TWT always removes and then replaces images during product importation, so there will be no old, unused image files in the media directory. (Contrary to the case when the standard Magento image removal is used.)
- The module imports all images but doesn't import textual data from the *service_media* table. (To bind data to product attributes, see the Binding Text Fields from Service_Media Table section.)

## 3.3 Main Class Descriptions

The main classes in the Teamwork Transfer module are:

- Teamwork_Transfer_IndexController: an "entry-point" class for the module's actions. The main actions are:
  - staging: precedes ECM requests
  - status: sets order status

- Teamwork_Transfer_Model_Transfer: an “entry-point” class for ECM request processing.
- Teamwork_Transfer_Model_Webstaging: creates weborder records in staging tables after Magento order creation. “Activate” is initial method, then Magento calls on “sales_order_save_after” event. (See etc/config.xml.)
- Teamwork_Transfer_Model_Status: changes order status.
- Teamwork_Transfer_Model_Media: operates “service_media” table to get product and category media data (images and text).
- Teamwork_Transfer_Model_Class_Attribute: imports or updates product attribute data.
- Teamwork_Transfer_Model_Class_Category: imports or updates product categories.
- Teamwork_Transfer_Model_Class_Item: imports or updates simple and configurable products.
- Teamwork_Transfer_Model_Class_Package: imports or updates product bundle products.
- Teamwork_Transfer_Model_Class_Price: imports or updates product prices.
- Teamwork_Transfer_Model_Class_Quantity: imports or updates stock data.
- Teamwork_Transfer_Model_Class_Synchronization: synchronizes category structure from staging tables to existing Magento category structure. The synchronization is turned off by default and usually used only once while module is installing (It should be enabled manually using the “isUse” property in “Teamwork_Transfer_Model_Class_Synchronization”.)
- Teamwork_Transfer_Helper_Data: used for rewrite mechanism.

## 3.4 Customization

It may occasionally be necessary to customize to the code. Some classes can be rewritten. The current module version uses a class rewrite mechanism different than the standard Magento rewrite. Rewrite rules are described in the <default><classes> section in etc/system.xml. This section contains pairs <name> - <reload>. The <reload> contains a class name which can be used instead of <name>. Class naming rules are the same as Magento, so all rewrite classes should be located in Model\Rewrite directory. This keeps all code (base + custom) in a single module and helps to add customization code without changing base code, which is useful for updating. To create any object of classes listed in <default><classes> section, call

```
$obj = Mage::helper('teamwork_transfer')-
>getClassByName({name}),
```

where {name} is a container from <default><classes> (for example “media”, “webstaging”),

instead of Mage::getModel(…) or Mage::getSingleton(…).

### 3.4.1 Binding Text Fields from Service_Media Table

The "service_media" table contains image and text product and category data. The Teamwork Service module contains the code to bind images from the "service_media" table to Magento products and categories, but doesn't contain the code to bind textual data. Usually this table keeps some text product attribute values like "description", "short_description", etc. To bind these records to Magento product attributes, rewrite "beforeAddData" method of "Teamwork_Transfer_Model_Class_Item" class. (To add "Item.php" to "Model/Reload" which should define "Teamwork_Transfer_Model_Reload_Item, extend the Teamwork_Transfer_Model_Class_Item" class). The following is an example of attaching the "description" attribute:

```
class Teamwork_Transfer_Model_Reload_Item extends
Teamwork_Transfer_Model_Class_Item
{
...
    protected function beforeAddData(&$productData, &$style, &$typeId,
&$item, &$visibile)
    {
        $txts = $this->_mediaModel
                    ->getMediaTexts($style['style_id'],
                                    'style',
                                    $this->_globalVars['channel_id']);
        if(!empty($txts))
        {
          if($visibile)
          {
              $productData['description'] = !empty($txts[{media_index}]) ?
$txts[{media_index}] : '';
          }
          else
          {
              $productData['description'] = '';
          }
        }
    }
...
}
```

where `{media_index}` is "media_index" column value of "service_media" table.

After adding the new code, restart the product import process. The best way is to resend the products using the Teamwork_Service API, which will update the staging tables and "ask" the Teamwork Transfer module to update all products data.

# 4. Staging Tables

In the current version of ecommerce integration, there are 55 staging tables. All of these tables begin with the prefix “service” when Magento’s internal prefix is empty.

## 4.1 Purpose

Staging tables are intermediaries between Teamwork Retail and an eCommerce website, containing all of the data that goes between them and allowing the data to be translated from one platform to another. Integration to most EC systems can be made using the staging tables.

## 4.2 Staging Table Workflow

Staging tables are between CHQ and the ecommerce website. Communication between the two is done in an XML format.

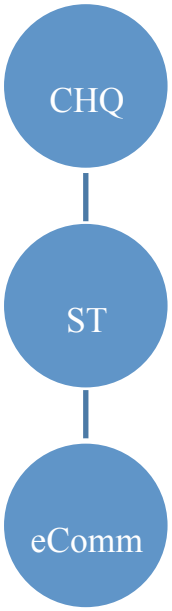
There are two concepts in using the tables:

- to pile up all the incoming information from the CHQ.
- to store all orders committed at the eCommerce website.

General provisions:

- As an identifier in staging tables, Globally Unique Identifier (guid) is being used.
- Staging tables terminology based on XML that comes from CHQ. Therefore some discrepancies with commonly used eCommerce terminology might be present.

Figure 2, Staging Table Workflow, illustrates how staging tables function in the system.




2. Staging Table Workflow

## 4.3 Staging Table Structure

Five groups of tables are classified in Teamwork Retail:

- Service tables
- Setting tables
- Catalog tables
- Control tables
- Web-order tables

**Note**

Some staging tables have been created for future use and are not currently included during data import.

### 4.3.1 Service Tables

This group includes two tables, service and service_channel.

service

This is a basic table from which the import process begins.

| | | |
|---|---|---|
| request_id | CHAR(36) | request identifier |
| rec_creation | TIMESTAMP | timestamp of Ecm request |
| channel_id | CHAR(36) | shows us what channel is used in the Ecm request |
| status | VARCHAR(255) | not used in current version |

| | | |
|---|---|---|
| type | VARCHAR(255) | not used in current version |
| chunk_group_count | INT(10) | internal counter is used for chunks with same type |
| chunk | INT(10) | counter increases this amount when a chunk comes |
| total_chunk | INT(10) | shows total chunk quantity |

**service_channel**

This table keeps information about all existing channels in CHQ. Note that one ECM request is related to one channel.

| | | |
|---|---|---|
| channel_id | CHAR(36) | channel identifier |
| channel_name | VARCHAR(100) | channel name |
| qty_location | CHAR(36) | send quantity for location with id, if empty then send company quantity |

**Note**

For proper functioning, your eCommerce platform must be stored with the same name as the channel name on which the request was sent.

### 4.3.2 Setting Tables

This group contains three tables: service_settings, service_setting_payment, and service_setting_shipping.

**service_settings**

This table can be populated with any setting information that comes from CHQ.

| | | |
|---|---|---|
| setting_name | VARCHAR(100) | setting name |
| setting_value | VARCHAR(100) | setting value |

**service_setting_payment**

This table stores all the information about the payment methods.

| | | |
|---|---|---|
| name | VARCHAR(255) | name of payment method |
| channel_id | CHAR(36) | channel identifier |
| description | TEXT | description |
| active | TINYINT(1) | condition of payment method |

**service_setting_shipping**

This table stores all the information about shipping methods.

| | | |
|---|---|---|
| name | VARCHAR(255) | name of shipping method |
| channel_id | CHAR(36) | channel identifier |
| description | TEXT | description |
| active | TINYINT(1) | condition of payment method |

### 4.3.3 Catalog Tables

These tables directly reflect the condition of EC content. These tables are filled in with CHQ XML content, and later the content transfers to EC. Review each table below.

The two tables below can be considered the main tables in this section: service_style and service_items. Both these tables are responsible for product creation.

One style may contain one or more items; for this reason we can build either a simple product, or a configurable one.

**service_style**

| | | |
|---|---|---|
| style_id | CHAR(36) | style identifier |
| request_id | CHAR(36) | request identifier |
| internal_id | INT(10) | internal eCommerce product identifier |
| no | VARCHAR(255) | style number |
| description {1…4}, ecommdescription | TEXT | description fields |
| ecomerce | VARCHAR(100) | ECommerce option defines availability of the style in eCommerce site; EC Offer - status Enabled; otherwise, status Disabled |
| dcss | CHAR(36) | department identifier |
| acss | CHAR(36) | alternate classification identifier |
| attributeset {1…3} | CHAR(36) | attribute name identifiers |
| brand | CHAR(36) | attribute value identifier for attribute brand |
| manufacturer | CHAR (36) | attribute value identifier for attribute manufacturer |
| customdate {1…6} | DATETIME | here can be stored any additional custom dates |
| customflag {1…6} | TINYINT(1) | here can be stored any additional custom bool statements |
| customlookup {1…12} | VARCHAR (255) | the value can be selected from customized user-defined list |
| customnumber {1…6} | DECIMAL(38,20) | additional custom decimal values |
| custominteger {1…6} | INT(10) | additional custom integer values |
| customtext {1…6} | VARCHAR(30) | short codes, etc |
| date_available | DATETIME | when the style is available for sale |
| inactive | TINYINT(1) | the style can be inactive (inactive = 1), in this case the style can be considered as deleted |

**service_items**

| | | |
|---|---|---|
| item_id | CHAR(36) | item identifier |
| request_id | CHAR(36) | request identifier |
| internal_id | CHAR(36) | internal eCommerce product identifier |
| style_id | CHAR(36) | style identifier |
| plu | VARCHAR(255) | Stock Keeping Unit (SKU) |
| ecomerce | VARCHAR(255) | the flag used for configurable products. EC Offer – adding enabled item to product; otherwise, disabled |

| | | |
|---|---|---|
| attribute{1…3}_id | CHAR(36) | identifier of value taken from style's attribute set |
| customdate {1…6} | DATETIME | for storing any additional custom dates |
| customflag {1…6} | TINYINT(1) | additional custom bool statements |
| customlookup {1…12} | VARCHAR (255) | value can be selected from customized user-defined list |
| customnumber {1…6} | DECIMAL(38,20) | additional custom decimal values |
| custominteger {1…6} | INT(10) | additional custom int values |
| customtext {1…6} | VARCHAR(30) | short codes etc… |

**Note**

The internal_id will be the same for both tables in the case of a simple product.

**Note**

Brand and manufacturer are used as attributes.

If you want to create a relationship between different products (Related, UpSells, CrossSells), you can use the **service_style_related** table.

**service_style_related**

| | | |
|---|---|---|
| style_id | CHAR(36) | style identifier |
| related_style_id | CHAR(36) | identifier for related style |
| related_style_type | VARCHAR(50) | type or relation (Related, UpSells, CrossSells, Custom, Unknown) |
| request_id | CHAR(36) | request identifier |

#### 4.3.3.1 Additional Tables

Additional tables for items and styles are service_style_category, service_style_channel, service_style_collection, service_item_category, service_item_channel, and service_item_collection.

These two category tables represent which category the style/item belongs to.

**service_style_category**

| | | |
|---|---|---|
| style_id | CHAR(36) | style identifier |
| category_id | CHAR(36) | category identifier |

**service_item_category**

| | | |
|---|---|---|
| item_id | CHAR(36) | item identifier |
| category_id | CHAR(36) | category identifier |
| request_id | CHAR(36) | request identifier |

#### 4.3.3.2 Channel Tables

The two channel tables show the product's channels. Each product can exist in different channels.

**service_style_channel**

| | | |
|---|---|---|
| style_id | CHAR(36) | style identifier |
| channel_id | CHAR(36) | channel identifier |

service_item_channel

| | | |
|---|---|---|
| item_id | CHAR(36) | item identifier |
| channel_id | CHAR(36) | channel identifier |

service_style_collection

| | | |
|---|---|---|
| style_id | CHAR(36) | style identifier |
| collection_id | CHAR(36) | collection identifier |

service_item_collection

| | | |
|---|---|---|
| item_id | CHAR(36) | item identifier |
| collection_id | CHAR(36) | collection identifier |

**Note**

This is the attribute value identifier for the "collection" attribute.

#### 4.3.3.3 Package Tables

The third product type is named package. In Magento, this type is called a “bundled product.” The product type can be created from various components and elements.

service_package

| | | |
|---|---|---|
| package_id | CHAR(36) | package product identifier |
| request_id | CHAR(36) | request identifier |
| internal_id | INT(10) | internal eCommerce identifier |
| description | VARCHAR(100) | description of product, used for product name |
| notes | TEXT | notation field, used for product information |

The next three tables, service_package_category, service_package_channel, and service_package_collection, correspond to the product type defined in the service package table above.

service_package_category

| | | |
|---|---|---|
| package_id | CHAR(36) | product identifier |
| category_id | CHAR(36) | category identifier |

service_package_channel

| | | |
|---|---|---|
| package_id | CHAR(36) | product identifier |
| channel_id | CHAR(36) | channel identifier |

service_package_collection

| | | |
|---|---|---|
| package_id | CHAR(36) | product identifier |
| collection_id | CHAR(36) | collection identifier |

As noted, packages can be created from various components and elements. The following two tables, service_package_component and service_package_component_element, provide information about these.

service_package_component

| | | |
|---|---|---|
| package_id | CHAR(36) | product identifier |
| comp_no | INT(10) | a number of component order |
| request_id | CHAR(36) | request identifier |
| description | TEXT | component title |
| allow_none | TINYINT(1) | true the component is not mandatory when the package is being sold |
| allow_multiple | TINYINT(1) | can be sold in quantities of more than one |

service_package_component_element

| | | |
|---|---|---|
| package_id | CHAR(36) | product identifier |
| no | INT(10) | component's element order number |
| item_id | CHAR(36) | item identifier |
| request_id | CHAR(36) | request identifier |
| price | DECIMAL(38,20) | element price |
| is_component_default | TINYINT(1) | Identifies the component default |
| quantity | INT(10) | available quantity |

The service_category table maintains all categories that have existed in CHQ.

service_category

| | | |
|---|---|---|
| *category_id* | **CHAR(36)** | *identifier of category* |
| *request_id* | **CHAR(36)** | *request identifier* |
| *channel_id* | **CHAR(36)** | *channel identifier* |
| *internal_id* | **INT(10)** | *internal category identifier* |
| *parent_id* | **CHAR(36)** | *parent category identifier* |
| *category_name* | **VARCHAR(255)** | *category name* |
| *description* | **VARCHAR(255)** | *category description* |
| *changed* | **DECIMAL(14,4)** | *shows in seconds when category was* |

| | | |
|---|---|---|
| | | changed |
| display_order | INT(10) | displays category order |
| is_active | TINYINT(1) | enabled / disabled category |

The service_attribute_set and service_attribute_value tables contain attributes by which the product can be further specified.

service_attribute_set

| | | |
|---|---|---|
| attribute_set_id | CHAR(36) | attribute identifier |
| request_id | CHAR(36) | request identifier |
| internal_id | INT(10) | internal eCommerce identifier of attribute |
| name | VARCHAR(30) | attribute name |

service_attribute_value

| | | |
|---|---|---|
| attribute_value_id | CHAR(36) | identifier of attribute value |
| attribute_set_id | CHAR(36) | identifier of attribute |
| attribute_value | VARCHAR(100) | attribute value |
| internal_id | INT(10) | internal eCommerce identifier of attribute value |

#### 4.3.3.4 Brand, Manufacturer and Collection Feature Tables

Note attributes are also created for the brand, manufacturer and collection elements.

service_brand

| | | |
|---|---|---|
| brand_id | CHAR(36) | brand identifier |
| name | VARCHAR(255) | attribute value |
| internal_id | INT(10) | internal value identifier |
| request_id | CHAR(36) | request identifier |

service_manufacturer

| | | |
|---|---|---|
| manufacturer_id | CHAR(36) | manufacturer identifier |
| name | VARCHAR(255) | attribute value |
| internal_id | INT(10) | internal value identifier |
| request_id | CHAR(36) | request identifier |

service_collection

| | | |
|---|---|---|
| collection_id | CHAR(36) | collection identifier |
| name | VARCHAR(255) | attribute value |
| internal_id | INT(10) | internal value identifier |
| description | VARCHAR(255) | description |

Collections can have their own category structure.

service_collection_category

| | | |
|---|---|---|
| collection_id | CHAR(36) | collection identifier |
| category_id | CHAR(36) | category identifier |

The service_fee table contains fee information. Here additional costs are saved, such as shipping or special offers not tied to a specific product.

service_fee

| | | |
|---|---|---|
| fee_id | CHAR(36) | fee identifier |
| code | VARCHAR(255) | fee code |
| description | VARCHAR(255) | fee description |
| alias | VARCHAR(128) | fee name, how it should appear on the site |
| item_level | TINYINT(1) | item level, if true then can be applied only on the item level |
| global_level | TINYINT(1) | global level, if true then can be applied on the whole order level |
| default_perc | DECIMAL(38,20) | default percent, if empty then not applicable |
| default_amount | DECIMAL(38,20) | default amount |

The next table is used for product quantity regulation.

service_inventory

| | | |
|---|---|---|
| item_id | CHAR(36) | item identifier |
| location_id | CHAR(36) | location identifier |
| channel_id | CHAR(36) | channel identifier |
| request_id | CHAR(36) | request identifier |
| quantity | INT(10) | product quantity |

The next table is used for pricing. We can support up to ten price levels associated with any reason, such as preferred pricing, etc.

service_price

| | | |
|---|---|---|
| item_id | CHAR(36) | item identifier |
| request_id | CHAR(36) | request identifier |
| price_level | TINYINT(1) | price level |
| price | DECIMAL(38,20) | item price |

The next table is used for tracking media content: **style**, **location**, **item**, **category**, **collection**, **packages.**

service_media

| | | |
|---|---|---|
| media_uri | CHAR(36) | media identifier |
| host_type | ENUM | style, location, item, category, collection, package - groupings where media can be used |
| host_id | CHAR(36) | host identifier |
| media_type | VARCHAR(100) | media type, can be any type of RichMedia proposed by cHQ: Thumbnails, LargeImages, LongDescription, Videos, VideoLinks |
| media_name | VARCHAR(100) | name for media |
| media_sub_type | VARCHAR(100) | type of media content |
| media_indexs | INT(10) | type identifier, can be used for specifying content |

#### 4.3.3.5 Tables for Future Implementation

Additional tables are **service_identifier** and **service_discount.** In the current version of extensions, these tables do not affect eCommerce platform content. Their use will be implemented in the near future.

service_identifier

Here you can store additional identifiers for your items. For example Universal Product Code (UPC) or Electronic Identification (eID)

| | | |
|---|---|---|
| identifier_id | VARCHAR(100) | additional identifier |
| request_id | CHAR(36) | request identifier |
| item_id | CHAR(36) | item identifier |
| idclass | TINYINT(1) | class of the identifiers, 0 - UPC, 1 - EID |
| value | VARCHAR(255) | identifier value |

service_discount

| | | |
|---|---|---|
| discount_id | CHAR(36) | discount identifier |
| code | VARCHAR(30) | code for discount |
| description | VARCHAR(100) | eComm alias - how it should appear on the site |
| type | TINYINT(1) | 0-line; 1-global |
| default_perc | DECIMAL(38,20) | default percent, if empty then not applicable |

#### 4.3.3.6 Tables for Store Use

The first two tables are used if you want to add locations to your eCommerce store.

**Note**

The location tables, while used for the retail store, are not used during importation from staging tables to the eCommerce platform. They can be used as desired.

service_location

| | | |
|---|---|---|
| location_id | CHAR(36) | location identifier |

| | | |
|---|---|---|
| code | VARCHAR(255) | location code |
| enabled | TINYINT(1) | enabled / disabled store for ecomm |
| name | VARCHAR(255) | location name |
| contact | VARCHAR(255) | location contact information |
| address {1-4} | VARCHAR(255) | location addresses |
| postal_code | VARCHAR(255) | location postal/zip code |
| city | VARCHAR(255) | location city |
| state | VARCHAR(255) | location state |
| country | VARCHAR(255) | location country |
| longitude | VARCHAR(255) | location longitude |
| latitude | VARCHAR(255) | location latitude |
| phone | VARCHAR(255) | location phone |
| fax | VARCHAR(255) | location fax |
| email | VARCHAR(255) | location email |
| home_page | VARCHAR(255) | location home page |
| alias | VARCHAR(255) | location alias |
| is_open | VARCHAR(255) | enabled / disabled store generally |
| location_price_group | VARCHAR(255) | price group for the location |
| customdate {1…6} | DATETIME | additional custom dates |
| customflag {1…6} | TINYINT(1) | additional custom bool statements |
| customlookup {1…12} | VARCHAR (255) | the value can be selected from customized user-defined list |
| customnumber {1…6} | DECIMAL(38,20) | additional custom decimal values |
| custominteger {1…6} | INT(10) | additional custom int values |
| customtext {1…6} | VARCHAR(30) | short codes etc… |

**service_location_schedule**

| | | |
|---|---|---|
| location_id | CHAR(36) | location identifier |
| open_time | DATETIME | store opens |
| close_time | DATETIME | store closes |
| day | TINYINT(1) | day number |
| closed | TINYINT(1) | day number when store close |

The following tables are used if you want to use department sections inside your store. DCSS = department/class/subclass identifier. ACSS - alternate classification.

**Note**

The department section tables, while used for the retail store, are not used during importation from staging tables to the eCommerce platform. They can be used as desired.

**service_dcss**

| | | |
|---|---|---|
| dcss_id | CHAR(36) | department classification (DCSS) identifier |
| request_id | CHAR(36) | request identifier |
| department_id | CHAR(36) | department identifier |
| class_id | CHAR(36) | class identifier |
| subclass1_id | CHAR(36) | class identifier |
| subclass2_id | CHAR(36) | class identifier |

### service_dcss_class

| | | |
|---|---|---|
| class_id | CHAR(36) | class identifier |
| request_id | CHAR(36) | request identifier |
| code | VARCHAR(255) | class code |
| name | VARCHAR(255) | class name |

### service_dcss_subclass1

| | | |
|---|---|---|
| subclass1_id | CHAR(36) | class identifier |
| request_id | CHAR(36) | request identifier |
| code | VARCHAR(255) | class code |
| name | VARCHAR(255) | class name |

### service_dcss_subclass2

| | | |
|---|---|---|
| subclass2_id | CHAR(36) | class identifier |
| request_id | CHAR(36) | request identifier |
| code | VARCHAR(255) | class code |
| name | VARCHAR(255) | class name |

### service_dcss_department

| | | |
|---|---|---|
| department_id | CHAR(36) | department identifier |
| request_id | CHAR(36) | request identifier |
| code | VARCHAR(255) | department code |
| name | VARCHAR(255) | department name |

### service_acss

| | | |
|---|---|---|
| acss_id | CHAR(36) | alternate classificaion (ACSS) |
| request_id | CHAR(36) | request identifier |
| level{1…4}_id | CHAR(36) | level identifier |

### service_acss_level1

| | | |
|---|---|---|
| level1_id | CHAR(36) | level identifier |
| request_id | CHAR(36) | request identifier |
| code | VARCHAR(255) | alternate classification code |
| name | VARCHAR(255) | alternate classification name |

### service_acss_level2

| | | |
|---|---|---|
| level2_id | CHAR(36) | level identifier |
| request_id | CHAR(36) | request identifier |
| code | VARCHAR(255) | alternate classification code |
| name | VARCHAR(255) | alternate classification name |

### service_acss_level3

| | | |
|---|---|---|
| level3_id | CHAR(36) | level identifier |

| request_id | CHAR(36) | request identifier |
|---|---|---|
| code | VARCHAR(255) | alternate classification code |
| name | VARCHAR(255) | alternate classification name |

service_acss_level4

| level4_id | CHAR(36) | level identifier |
|---|---|---|
| request_id | CHAR(36) | request identifier |
| code | VARCHAR(255) | alternate classification code |
| name | VARCHAR(255) | alternate classification name |

### 4.3.4 Control Tables

These tables are used for situations when you want to control some aspect of the process directly from CHQ. The current Teamwork Retail version supports only one option: status change for web-orders.

Three tables are provided: **service_status**, **service_status_items**, and **service_status_shipping.**

service_status

| PackageId | CHAR(36) | package xml identifier |
|---|---|---|
| WebOrderId | CHAR(36) | order identifier |
| Status | VARCHAR(50) | new order items status |

service_status_items

| ItemId | CHAR(36) | item identifier |
|---|---|---|
| PackageId | CHAR(36) | package xml identifier |
| Qty | INT(10) | items quantity, which status will be changed |
| internal_id | INT(10) | internal identifier |

The table is used for items for which shipping status is desired.

service_status_shipping

| ShippingInformation Id | CHAR(36) | shipping identifier |
|---|---|---|
| PackageId | CHAR(36) | package xml identifier |
| Carrier | VARCHAR(50) | carrier of shipping method |
| ShippingMethod | VARCHAR(50) | shipping method |
| TrackingNo | VARCHAR(100) | tracking number |
| Estimate | VARCHAR(100) | estimation |
| Description | VARCHAR(100) | additional information |

### 4.3.5 Web-order Tables

The following tables work in the other direction, and are populated by eCommerce order information during the process of fulfilling orders received.

service_weborder

| | | |
|---|---|---|
| WebOrderId | CHAR(36) | web-order identifier |
| EComChannelId | CHAR(36) | channel identifier |
| EComShippingMethod | VARCHAR(50) | Shipping method |
| OrderNo | VARCHAR(50) | order number |
| OrderDate | DATETIME | order date |
| EComCustomerId | VARCHAR(128) | customer identifier |
| {Bill,Ship} FirstName | VARCHAR(50) | first name |
| {Bill,Ship} LastName | VARCHAR(50) | last name |
| {Bill,Ship} MiddleName | VARCHAR(50) | middle name |
| {Bill,Ship} Gender | TINYINT(1) | gender |
| {Bill,Ship} Birthday | DATETIME | birthday |
| {Bill,Ship} Email | VARCHAR(50) | email |
| {Bill,Ship} Phone | VARCHAR(50) | phone |
| {Bill,Ship} MobilePhone | VARCHAR(50) | mobile |
| {Bill,Ship} Company | VARCHAR(50) | company |
| {Bill,Ship} Address{1,2} | VARCHAR(128) | address |
| {Bill,Ship} City | VARCHAR(128) | city |
| {Bill,Ship} Country | VARCHAR(50) | country |
| {Bill,Ship} PostalCode | VARCHAR(50) | postal / zip code |
| CustomDate {1…6} | DATETIME | additional custom dates |
| CustomFlag {1…6} | TINYINT(1) | additional custom bool statements |
| CustomLookupValue{1…6} | VARCHAR (255) | the value can be selected from customized user-defined list |
| CustomNumber {1…6} | DECIMAL(38,20) | additional custom decimal values |
| CustomInteger {1…6} | INT(10) | additional custom int values |
| CustomText {1…6} | VARCHAR(30) | short codes etc… |
| Instruction | Text | instructions |

service_weborder_discount_reason

| | | |
|---|---|---|
| WebOrderId | CHAR(36) | weborder identifier |
| GlobalDiscountReasonId | CHAR(36) | discount identifier |
| GlobalDiscountAmount | DECIMAL(38,20) | discount amount |

service_weborder_fee

Shipping information is stored here. For shipping the quantity is always equal to 1.

| | | |
|---|---|---|
| FeeId | CHAR(36) | global fee identifier |
| WebOrderId | CHAR(36) | web-order identifier |
| TaxAmount | DECIMAL(38,20) | fee tax |
| UnitPrice | DECIMAL(38,20) | whole fee price |
| Qty | INT(10) | quantity |

### service_weborder_item

| | | |
|---|---|---|
| WebOrderItemId | CHAR(36) | weborder item identifier |
| WebOrderItemsGroupId | CHAR(36) | weborder item group identifier |
| WebOrderId | CHAR(36) | weborder identifier |
| ItemId | CHAR(36) | item identifier |
| OrderQty | DECIMAL(38,20) | quantity for ordered items |
| UnitPrice | DECIMAL(38,20) | price for an item |
| LineTaxAmount | DECIMAL(38,20) | tax amount |
| TrackingNo | VARCHAR(50) | tracking number |
| LineNo | INT(10) | consecutive number |
| CustomDate {1…6} | DATETIME | additional custom dates |
| CustomFlag {1…6} | TINYINT(1) | additional custom bool statements |
| CustomLookupValue{1…6} | VARCHAR(255) | the value can be selected from customized user-defined list |
| CustomNumber {1…6} | DECIMAL(38,20) | additional custom decimal values |
| CustomInteger {1…6} | INT(10) | additional custom int values |
| CustomText {1…6} | VARCHAR(30) | short codes etc… |
| Notes | TEXT | weborder notation |

### service_weborder_item_fee

| | | |
|---|---|---|
| WebOrderItemId | CHAR(36) | Item identifier |
| FeeId | CHAR(36) | local fee identifier |
| UnitPrice | DECIMAL(38,20) | unit fee |
| TaxAmount | DECIMAL(38,20) | tax of local fee |
| Qty | INT(10) | fee unit quantity |

### service_weborder_item_line_discount

| | | |
|---|---|---|
| WebOrderItemId | CHAR(36) | web-order item identifier |
| LineDiscountReasonId | CHAR(36) | discount identifier |
| LineDiscountAmount | DECIMAL(38,20) | discount amount |

### service_weborder_payment

| | | |
|---|---|---|
| WebOrderPaymentId | CHAR(36) | payment identifier |
| WebOrderId | CHAR(36) | order identifier |
| CardType | VARCHAR(50) | card type |
| EComPaymentMethod | VARCHAR(50) | payment method |
| AccountNumber | VARCHAR(38) | account number |
| PaymentAmount | DECIMAL(38,20) | payment amount |
| CardExpMonth | TINYINT(2) | expiration month |
| CardExpYear | SMALLINT(4) | expiration year |
| MerchantId | VARCHAR(128) | merchant identifier |
| CardOrderId | VARCHAR(255) | card order identifier |
| ReferenceNum | VARCHAR(255) | reference number |
| TransactionId | VARCHAR(255) | transaction identifier |
| ListOrder | INT(10) | consecutive payment number |

| | | |
|---|---|---|
| CardholderFirstName | VARCHAR(128) | cardholder first name |
| CardholderLastName | VARCHAR(128) | cardholder last name |
| CardholderAddress {1-2} | VARCHAR(128) | cardholder address |
| CardholderCity | VARCHAR(128) | cardholder city |
| CardholderState | VARCHAR(128) | cardholder state |
| CardholderCountryCode | VARCHAR(128) | cardholder country code |
| CardholderPostalCode | VARCHAR(128) | cardholder postal code |
| LoyaltyRewardPointAmount | INT(10) | loyalty reward point amount |

# 5. Appendix A: Service Module Structure

Structure of the module meets the typical structure of Magento modules:

- the all configuration files are in “etc” directory;
- the all installation scripts are in “sql” directory;
- the all php classes are in separate files located in the following directories: “controllers”; “Helper”; “Model”. The name of each class contains path to their file obeying the Magento rules.

Directory structure:

```
Teamwork
    Service
        controllers
            IndexController.php
        etc
            api.xml
            config.xml
        Helper
            Data.php
        Model
            Adapter
                Db.php
            Api.php
            Ecm.php
            Mapping.php
            Service.php
            Settings.php
            Status.php
            Version.php
            Weborder.php
        sql
            service_setup
                mysql4-install-2.1.9.php
                mysql4-upgrade-2.1.9-2.1.10.php
                mysql4-upgrade-2.1.10-2.1.11.php
                ...
```

Description of the files:

`Teamwork/Service/etc/config.xml`
The file contains module configuration.

`Teamwork/Service/etc/api.xml`
The file contains module configuration related to XML-RPC. Contains mapping XML-RPC requests – module actions.

`Teamwork/Service/controllers/IndexController.php`

The file contains Teamwork_Service_IndexController class and represents some debug actions to simulate XML-RPC request.

`Teamwork/Service/Helper/Data.php`
The file contains Teamwork_`Service`_Helper_Data class. The helper contains methods which uses curl to call Transfer module.

`Teamwork/Service/Model/Adapter/Db.php`
The file contains `Teamwork_Service_Model_Adapter_Db` class and contains methods designed to simplify database operations. The all classes database uses encapsulates an object of this class to get database access.

`Teamwork/Service/Model/Api.php`
The file contains `Teamwork_Service_Model_Api` class and represents the entry point of the all module actions (see Teamwork/Service/etc/api.xml to get methods mapping).

`Teamwork/Service/Model/Ecm.php`
The file contains Teamwork_Service_Model_Ecm class and designed to return ECM requests' statuses.

`Teamwork/Service/Model/Mapping.php`
The file contains Teamwork_Service_Model_Mapping class and designed to exchange of mapping data.

`Teamwork/Service/Model/Service.php`
The file contains Teamwork_Service_Model_Service class and designed to operate ECM request.

`Teamwork/Service/Model/Settings.php`
The file contains Teamwork_Service_Model_Settings class and designed to exchange of settings data.

`Teamwork/Service/Model/Status.php`
The file contains Teamwork_Service_Model_Status class and designed to set orders' statuses (fills "service_status" and "service_status_items" staging tables and calls Transfer module to change orders' statuses).

`Teamwork/Service/Model/Version.php`
The file contains Teamwork_Service_Model_Version class and designed to return module version.

`Teamwork/Service/Model/Weborder.php`
The file contains Teamwork_Service_Model_Weborder class and designed to return weborder's (the all "fresh" data from "service_weborder" staging table which fills by Transfer module).

```
Teamwork/Service/sql/service_setup/mysql4-install-2.1.9.php,
Teamwork/Service/sql/service_setup/mysql4-upgrade-2.1.9-2.1.10.php,
...
```
The files contains installation code, Magento uses while module installation/updating. The code from these files designed to create and update the all needed staging tables.

# 6. Appendix B: Transfer Module Structure

Structure of the module meets the typical structure of Magento modules:

- all configuration files are in “etc” directory;
- all installation scripts are in “sql” directory (there are no installation scripts here for current module state);
- the all php classes are in separate files located in the following directories: “controllers”, “Helper”, “Model”. The name of each class contains path to their file obeying the Magento rules.

Directory structure:

```
Teamwork
    Transfer
        controllers
            IndexController.php
        etc
            config.xml
        Helper
            Data.php
            Log.php
        Model
            Class
                Attribute.php
                Category.php
                Item.php
                Label.php
                Package.php
                Price.php
                Quantity.php
                Synchronization.php
            Abstract.php
            Media.php
            Status.php
            Transfer.php
            Webstaging.php
        sql
            transfer_setup
```

Description of the files:

`Teamwork/Transfer/etc/config.xml`
The file contains module configuration.

`Teamwork/Transfer/controllers/IndexController.php`
The file contains Teamwork_Transfer_IndexController class and represents the entry point of the all module actions. The controller extends “adminhtml controller” to perform all the actions as

an administrator. The controller rewrote “preDispatch” method with some special code to allow the all administrator actions without admin user validation.

`Teamwork/Transfer/Helper/Data.php`
The file contains Teamwork_Transfer_Helper_Data class with empty body for current module state. Can be used for translation functions if add special configuration section to config.xml or for another purpose.

`Teamwork/Transfer/Helper/Log.php`
The file contains Teamwork_Transfer_Helper_Log class. It is designed to simplify logging to separate file from different module place. This helper can add to log textual info and also parsed information from exception object.

`Teamwork/Transfer/Model/Abstract.php`
The file contains Teamwork_Transfer_Model_Abstract class. It contains the code for storing for ECM reporting and logging to external file, using module’s helper, the all error and warning messages occurred while ECM processing.

`Teamwork/Transfer/Model/Transfer.php`
The file contains Teamwork_Transfer_Model_Transfer class. It represents an entry point for ECM request processing mechanism. The “run” is initial method Index controller calls to start ECM processing. It extends Teamwork_Transfer_Model_Abstract class and contains ECM error and warning reporting mechanism. The all classes from Teamwork/Transfer/Model/Class directory extends this class to use this mechanism.

`Teamwork/Transfer/Model/Media.php`
The file contains Teamwork_Transfer_Model_Media class. It is designed to operate “service_media” table to get product and categories and ‘rich media’ data (images and textual data such as descriptions etc.).

`Teamwork/Transfer/Model/Status.php`
The file contains Teamwork_Transfer_Model_Status class. It is designed to change orders’ statuses.

`Teamwork/Transfer/Model/Webstaging.php`
The file contains Teamwork_Transfer_Model_Webstaging class. It is designed to create weborders’ records in staging tables after Magento orders creation. The “activate” is initial method Magento calls on “sales_order_save_after” event (see etc/config.xml).

`Teamwork/Transfer/Model/Class/Attribute.php`
The file contains Teamwork_Transfer_Model_Class_Attribute class. It is designed to import/update product attributes’ data. The class extends Teamwork_Transfer_Model_Transfer class.

`Teamwork/Transfer/Model/Class/Category.php`
The file contains Teamwork_Transfer_Model_Class_Category class. It is designed to import/update product categories. The class extends Teamwork_Transfer_Model_Transfer class.

`Teamwork/Transfer/Model/Class/Item.php`

The file contains Teamwork_Transfer_Model_Class_Item class. It is designed to import/update “simple” and “configurable” products. The class extends Teamwork_Transfer_Model_Transfer class.

`Teamwork/Transfer/Model/Class/Label.php`
The file contains Teamwork_Transfer_Model_Class_Label class. It is designed to update labels of configurable attributes. The class extends Teamwork_Transfer_Model_Transfer class.

`Teamwork/Transfer/Model/Class/Package.php`
The file contains Teamwork_Transfer_Model_Class_Package class. It is designed to import/update “boundle” products. The class extends Teamwork_Transfer_Model_Transfer class.

`Teamwork/Transfer/Model/Class/Price.php`
The file contains Teamwork_Transfer_Model_Class_Price class. It is designed to import/update product prices. The class extends Teamwork_Transfer_Model_Transfer class.

`Teamwork/Transfer/Model/Class/Quantity.php`
The file contains Teamwork_Transfer_Model_Class_Quantity class. It is designed to import/update stock data. The class extends Teamwork_Transfer_Model_Transfer class.

`Teamwork/Transfer/Model/Class/Synchronization.php`
The file contains Teamwork_Transfer_Model_Class_Synchronization class. It is designed to synchronize category structure from staging tables to existing Magento category structure. The synchronization is turned off by default and usually used only once while module installing to working Magento shop (can be enabled manually using “isUse” property). The class extends Teamwork_Transfer_Model_Transfer class.

# 7.Index to Staging Tables