KHAOS CONTROL

sales@khaoscontrol.com
www.khaoscontrol.com
0845 257 5111

Khaos Control
Web Integration Guide v7.8d

Table of Contents

Introduction

The Khaos Control Web Services are a generic set of services that are, by design, not specific to any particular integration. There are many operations and functions available via the web services, although it is not mandatory to utilise all available feeds and operations.

Included in the Web Integration Pack is our Recommended Web Integration Process document that outlines an ideal integration scenario; Khaos Control being responsible for holding and maintaining all details of customer accounts, stock, stock categorisation, etc that may be required by the web site, and the web services providing these upon request with the website maintaining and updating a cache of these details.

This document explains how to integrate Khaos Control with an external system (primarily a website) and exchange information between Khaos Control and the external system.

Why XML?

Our web integration routines utilize a number of standard XML file formats (samples are included with this guide). We use XML because:

- XML is a flexible file format – provided the software reading the file is properly XML compliant, it is usually very easy to add new fields to the file format without causing any problems with older software. The older software should simply ignore the fields it does not need to read.
- XML files are generally also Unicode compatible (i.e. they can include non-UK letters and characters without problems) although Unicode is only supported within specific elements of Khaos Control. Please contact KCSL to discuss your Unicode requirements in detail if your website(s) will require this compatibility.
- Validation tools, that check whether your XML file is valid or not, are widely available – at the most basic level, Internet Explorer can load an XML file and tell you whether there are any serious problems with the file or not.

Data Exchange Overview

We exchange data using “SOAP Web services”. The Simple Object Access Protocol (SOAP) is an XML based protocol that lets applications exchange information over HTTP or HTTPS. It is a platform-independent method of exchanging information across the Internet, which uses XML and HTTP to complete the exchange. KCSL has a standard web service module designed to exchange data to/from Khaos Control, which is described in more detail below.

This Khaos web service (KhaosWS) will be installed on one of the client’s machines that also has Microsoft IIS installed and that can access the Khaos Control database on the client’s LAN. KhaosWS can be configured to run on any port designation that may be required. The client’s IT support company should handle the configuration any router/firewall issues relating to the web server, as KCSL are not network or security specialists We will install the web service using a Self-Signed Certificate so your web service will be accessible using a HTTPS connection as standard

KhaosWS is a standard interface and cannot be customised to an individual client’s specification.

Server Resources

Running a web service on your server will add extra load and could require more resources to be made available depending on the usage of your website. As a general rule, we recommend that your server be set up as a 10+ user system in line with the information contained in the Khaos Control System Requirements document (if you do not have a copy of this then please contact us to request

one). Server load can also be reduced by following the advice in the “Important Notes about Web Service Overuse” section below.

Important Note about Web Service Overuse

Although the web service is an efficient and effective way of integrating to Khaos Control, it can lead to problems if not planned correctly. Our clients have encountered problems before where web developers have taken short-cuts in their planning phases, leading to overuse of the web service. This can result in degradation of the client’s database server performance. Please bear in mind the following when planning your routines.

Stock

When synchronising stock, please be aware that potentially there can be an unlimited number of users visiting the website. If you check stock levels every time a stock item is viewed, this will result in having too many web service calls which will in turn slow the web service down and clog the log file up making it hard to ascertain where a problem lies as and when they arise. Likewise getting stock updates every second should not be a route that is taken either. We suggest getting stock levels every 30 minutes and storing them on the web server for product pages etc, and then separate ad-hoc checks when appropriate (on basket checkout, for example) to check “just-in-time” stock levels for the items on the customer’s order.

Importing Orders

Sometimes orders will fail to import due to stock errors or data issues. When this happens, developers tend to set up a routine to try and import the order again. This is both safe to do and advisable, however, if an order is failing to import due a stock error, it is going to keep failing when the “re-try routine” is called. We advise you only try importing an order a certain number of times and if it still won’t import at that point, notify a website administrator. This again will save the web service from being used when it doesn’t need to be and will stop the log files clogging up.

Available Methods in the Web Service

Below is a list of the methods KhaosWS supports.

Import (into Khaos Control)	Export (from Khaos Control)
Sales Order(s)	[1] Sales Order Status [Basic]
Customer Detail	[1] Sales Order Status [Extended]
Catalogue Request(s)	[1] Sale Sources
Stock Status	[1] Customer Login Verification
Price Lists	[2] Customers that have changed List
	[2] Entries [3] that have been deleted List
	[2][1] Customer Detail
	[1] Country List
	[1] Company Class List
	[2] Stock Items that have changed List
	[2][1] Stock Item Detail
	[2][1] Stock Item Detail - Compressed
	[1] Stock Status
	[1] Stock Status - Compressed
	[1][2] Web Categories
	[1][2] Web Categories - Compressed
	[1][2] Web Categories - Ex2
	[1] Price List(s)
	[1] Manufacturer
	[1] Version of kOS/KC
	[1] Keycodes

[1] Consider requesting "as and when" the data is required on an ad-hoc basis.

[2] Consider requesting from the web service as an update routine that populates your local web tables.

[3] Currently 'Entries' only refers to companies.

Sales Order Data

Import

```
function ImportOrders(Orders: WideString): TOrderImportResult;
```

This method expects one parameter:

`Orders` **– XML data as defined in the file KSDXMLImportFormat.xml**

Note: Certain payment provider integrations require specific tags to be present in the supplied XML. Please see the accompanying PSP Required Fields document for further information.

Return Results:

`TOrderImportResult =`

`property ImportedCount: Integer` **– Total count of all sales orders that were successfully imported into Khaos Control.**

`property OrderImport: TOrderImportArray` **– An array of** `TOrderImport`

`property OrderSkipped: TOrderSkippedArray` **– An array of** `TOrderSkipped`

`TOrderImport =`

`property SalesOrderCode: String`

property AssociatedRef: String – The web site's order reference
property Msg: String – Contains messages pertaining to the import process
property URN: String – Customer Unique Reference Number (alpha)
property OtherRef: String – The web site's customer reference

TOrderSkipped =
property OrderRef: WideString – The web site's order reference
property SkipReason: WideString – Text that relates to why the order could not be imported.

See also Matching Customer Records, below.
See also Khaos Control – Notes / VAT, below for information on how Khaos will use the values you send through.
See also Dealing with Discounts and Keycodes, below.

Export

```
function ExportOrderStatus(): WideString;
```

This method will export the status for all web orders (i.e. those with a valid web / associated reference) in Khaos Control that have been marked as Issued in the past 48 hours, plus all of the order that are not marked as issued and are therefore still being processed.

Return Results:
XML data as defined in the file KSDXMLOrderStatus.xml

Export

```
function ExportOrderStatusEx(IDs: WideString; MappingType: Integer;
WebOnly, ConfirmedOnly, IncludeTracking, IncludeCancelled: Boolean;
DateFrom: TXSDateTime; DateTo: TXSDateTime; InvoiceStageIDs: WideString;
ShowItems: Boolean): WideString;
```

This method expects up to ten parameters:

IDs – (optional*) The value should be passed as a comma separated string of values. The values correspond to MappingType.
*This field is only optional when only the DateFrom and DateTo parameters have been set.

MappingType – (mandatory) Determines what "IDs" will be looked-up against in Khaos Control.

Values are:
1: 'COMPANY_CODE' - Khaos Control's customer reference (URN)
2: 'OTHER_REF' – the web site's customer reference
3: 'SORDER_CODE' - Khaos Control's order reference
4: 'ASSOCIATED_REF' – the web site's order reference
If no ID(s) are defined, set this to 1.

WebOnly – (optional) If specified determines whether or not orders with a web reference should be exported. Default value is 0 (false).

ConfirmedOnly – (optional) If specified determines whether or not only those orders marked as confirmed should be exported. Default value is 0 (false).

IncludeTracking – (optional) Used in conjunction with below date filters. If specified determines whether or not to include sales orders with tracking changes within the specified date range. Default value is 0 (false).

IncludeCancelled – (optional) If specified determines whether or not orders marked as cancelled (deleted) should be exported. Default value is 0 (false). To show only cancelled orders, set to -1 and use in combination with an InvoiceStageID of 1 (cancelled invoices).

DateFrom – (optional) If specified orders will be exported from this date and time. If omitted, results will not be filtered by date and time.

DateTo – (optional) If specified orders will be exported to this date and time. If omitted, results will not be filtered by date and time.

InvoiceStageIDs – (optional) If specified limits result set to those sales orders with invoices in the specified stage(s). Stage ID values are defined in KSDXMLOrderStatusEx.xml. Default is to return all stages.

ShowItems – determines whether the XML feed will include Invoice Item / Box (if invoices are being packed) data within the XML (-1) or not (0).

You can use this function in conjunction with **ExportOrders**, to display a list of historic orders and then display the detail for the one clicked on by the user.

Return Results:

XML data as defined in the file KSDXMLOrderStatusEx.xml

Export

```
function ExportOrders(OrderCode: WideString; MappingType: Integer): WideString;
```

This method expects two parameters:

OrderCode – The value should be passed as a comma separated string of values. The values correspond to MappingType.

MappingType – Determines what "IDs" will be looked-up against in Khaos Control.

Values are:

1: 'SORDER_CODE' – Khaos Control's order reference
2: 'ASSOCIATED_REF' – The web site's order reference

You can use this function in conjunction with **ExportOrderStatusEx**, to display a list of historic orders and then display the detail for the one clicked on by the user.

Return Results:

XML data as defined in the file KSDXMLImportFormat.xml

Export

```
function GetSaleSource(): TDataArray;
```

This method will export a list of all the Sales Source's set up in Khaos Control.

Return Results:

TDataArray – An array of WideString, contains the available Sales Sources

Customer Data

Export

```
function Login(UserName, UserPassword: WideString): WideString;
```

This method expects two parameters:

UserName – User name as defined in the Web Name field of the customer record in Khaos Control.

UserPassword – User password as defined in the Pwd field on the customer record in Khaos Control.

The function will look for the username and check the resulting record's password with the password supplied. If there is a correlation then the customer details will be returned, otherwise the return value

will be blank.

Return Results:
XML data as defined in the file KSDXMLCustomer.xml

Export

```
function GetCompanyList(LastUpdated: DateTime; CompanyClassList:
WideString): TCompanyList;
```

This method expects the following parameters:

`LastUpdated` – The date that the requested companies were last updated. This means that only companies that have been updated since this specified date will be sent in the response.

`CompanyClassList` – (optional) If specified only customers from the Company Class(es) specified will be exported out of Khaos Control. The value should be passed as a comma separated string of company classes.

Return Results:

```
TCompanyList =
      property Count: Integer – Total count of all company codes.
      property List: TCompanyArray – An array of TCompany
TCompany =
      property CompanyCode: WideString
      property OtherRef: WideString – Other reference for the company code.
      property LastUpdated: TDateTime – The date the company was last updated.
```

Note: The result should be used in conjunction with the ExportCompany method below. Web services have a tendency to sometimes cut-off or stop responding or timeout with large data transfers this method allows for a small data packet contain the company codes that have changed so that they can be passed in chunks to the ExportCompany method.

Export

```
function ExportCompany(CompanyCode: WideString; MappingType: Integer;
LastUpdated: TXSDateTime): WideString; stdcall;
```

This method expects a variable number of parameters:

`CompanyCode` – (optional*) If specified these customers will be exported out of Khaos Control. The value should be passed as a comma separated string of codes. The values do not have to be 'company code' – see MappingType next.

`MappingType` – Determines what "CompanyCode" will be looked-up against the Khaos Control stock table. Values are:

1: 'COMPANY_CODE' – main company code (URN), which is visible to the user (DEFAULT)
2: 'OTHER_REF' – the web site's customer reference
3: 'USER_NAME' – the web site's username for the customer, as stored in the 'Web Name' field
4: 'COMPANY_NAME' – on the customer form as the main Company Name
5: 'SSOURCE_DESC' – on the Sales Source defined against the customer

It is recommended that web developers use the USER_NAME parameter to pull company records from Khaos Control based on the WEB_USER value that was passed down in the original ImportCompany call for the customer.

`LastUpdated` – (optional*) If specified, all customer records that have been updated (changed in Khaos Control) since this chosen date will be exported.

* If both CompanyCode and LastUpdated parameters are specified, the CompanyCode parameter will take precedence.

Return Results:

XML data as defined in the file KSDXMLCustomer.xml – details of customers and their properties.

Note: Also see **GetCompanyList** above and the corresponding note.

Export

```
function GetCompanyInfo(CompanyCode: WideString; MappingType: Integer): TCompanyInfoArray;
```

This method expects two parameters:

CompanyCode – List of customers to be exported out of Khaos Control. The value should be passed as a comma separated string of codes. The values do not have to be 'company code' – see MappingType next.

MappingType – Determines what "CompanyCode" will be looked-up against the Khaos Control stock table. Values are:

1: 'COMPANY_CODE' – main company code (URN), which is visible to the user (DEFAULT)
2: 'OTHER_REF' – the web site's customer reference
3: 'USER_NAME' – the web site's username for the customer, as stored in the 'Web Name' field
4: 'COMPANY_NAME' – on the customer form as the main Company Name
5: 'SSOURCE_DESC' – on the Sales Source defined against the customer

Return Results:

TCustomerImportResult – An array of TCompanyInfo

TCompanyInfo =

property CompanyCode: String
property CompanyName: String
property OtherRef: String – Web site's customer reference
property HideCompany: Int – Shows whether a customer is anonymised
property HoldData: Int – Shows whether a customer is held
property CreditLimit: Currency – Credit limit in the customer's currency
property CreditBalance: Currency – Balance in the customer's currency which includes credit notes and payments as well as orders which are not yet issued (finalised) and orders which are. **Note:** both unallocated credit and unpaid invoices can result in a positive number. It is not recommended to use this field unless you are familiar with the functionality.
property CreditOverdue: Currency – Balance of unpaid issued invoices which have remained unpaid for more than the customer's terms length, in the customer's currency. In this case, a positive number means money the customer owes.
property CreditPending: Currency – Balance of only orders which are not yet issued (finalised), in the customer's currency. In this case, a positive number means money the customer is about to owe.
property CreditUnallocated: Currency – Unallocated payment and credit note balance, in the customer's currency. In this case, a positive number means money the customer has paid (or is credited), that has not been used to pay an invoice yet. Note this may include payments that were made for invoices not yet issued.
property OutstandingBalance: Currency – Equivalent to CreditOverdue – CreditUnalloacted. This is in the customer's currency. A positive number means money the customer owes, and a negative number means money the customer has paid (or is credited). **This property is only available in webservice version 8.173 onwards.**
property CurrencyCode: String – The code of the customer's currency, e.g. 'GBP'.
property TermsLength: Integer – Number of day's terms
property TermsType: Integer – From month end of invoice
property TermsDesc: String – Descriptive text relating to customer's credit terms

property CreditStop: Boolean – Is the customer on stop
property OrdersHeld: Boolean – Is the customer on order held
property PTypeID: Integer – Default payment type for customer 1 – cash, 2 – credit card, 3 - account
property Proforma: Boolean – Is this a proforma customer?
property Supplier: Boolean – Is this a supplier record or customer record?
property ECCompany: Boolean – Is the customer EC?
property PaysVAT: Boolean – Is the customer exempt from paying VAT for a "special" reason, e.g. they are a charity
property POCodeRequired: Boolean – Does this customer need a PO reference against their orders
property PriceLists: TDataArray – An array of WideString, containing the customer's available Price Lists

Export

```
function GetDeletedEntries(DateFrom: DateTime; DateTo: DateTime;
DocumentType: Integer; ObjectID: WideString): WideString;
```

This method expects three parameters:

DateFrom – (required) Used along with the DateTo to specify a date range; entries deleted on a date within this date range will be returned.
DateTo – (required) Used along with the DateFrom to specify a date range; entries deleted on a date within this date range will be returned.

DocumentType – (required) Determines what type of entry to look for. Possible values are:
5: Company entries.

ObjectID – (optional) (Not currently in use) This option is reserved for future development.

Return Results: See KSDXMLDeletedEntries.xml in the XML Examples folder.

NOTE: When an ADDITIONAL_REF is returned, this indicates the deleted entry was merged into another entry. Hence the applicable export, e.g. ExportCompany, should be called using the ADDITIONAL_REF as the ID to ensure that the record that was merged into is updated.

Import

```
function ImportCompany(Customers: WideString; Method: Integer):
TCustomerImportResult;
```

This method expects two parameters:
Customers – XML data as defined in the file KSDXMLCustomer.xml
Method – An integer value specifying the update method to be used. Possible values are:
0: (default action) create a new customer if one doesn't exist or update an existing record if one is found
1: create a new customer if *no* match is found and do nothing if a match *is* found
2: update existing records without creating new customers
3: force creation of a new customer without checking for duplicates **(this is not recommended unless you are certain the supplied COMPANY_CODE / URN is unique)**

Return Results:
TCustomerImportResult =
property ImportedCount: Integer – Total count of all newly generated customers.

property CustomerImport: TCustomerImportArray – An array of TCustomerImport

property CustomerSkipped: TCustomerSkippedArray – An array of TCustomerSkipped

TCustomerImport =

property URN: String – Customer Unique Reference Number (alpha), as created by Khaos Control

property Action: String – This returns as "New" if the import created a new customer and "Update" if the import updated an existing customer (changed in Khaos Control)

property OtherRef: String – Web site's customer reference

property PassedReference: String – The value of PASSED_REFERENCE as provided in the call to ImportCompany, which may be utilised to match requested and returned records

TCustomerSkipped =

property CustomerRef: String – Web site's customer reference

property SkipReason: String – Text that relates to the reason why this customer record was not imported

property PassedReference: String – The value of PASSED_REFERENCE as provided in the call to ImportCompany, which may be utilised to match requested and returned records

NOTE: When importing customers into Khaos Control, the XML file allows you to pass through ID's. These should only be used in specific circumstances and when discussed with Khaos Control Solutions.

See also Matching Customer Records, below.

Import

```
procedure ImportCatalogueRequest(CatRequest: WideString);
```

This method expects one parameter:

CatRequest – XML data from our KSDXMLCatRequest.xml file.

The XML will be imported as Catalogue Request(s) in Khaos Control.

NOTE: The ImportCatalogueRequest operation does not return a result.

Export

```
function GetCountryList(): TCountryList;
```

This method will export a list of all countries set up in Khaos Control.

Return Results:

TCountryList =

property Count: Integer – Total count of all countries.

property List: TCountryArray – An array of TCountry

TCountry =

property CountryName: String

property CountryCode2: String

property CountryCode3: String

property EUMember: Boolean – Returns true if the country is part of the EU.

property PaysVAT: Boolean – Returns true if the country pays VAT.

property CurrencyName: String

property CurrencyCode: String

property Zone: String

Export

```
function GetCompanyClass(CompanyClassType: Integer): TDataArray;
```

This method will export a list of all the Company Classes set up in Khaos Control.

This method expects one parameter:
`CompanyClassType` – (optional) An integer value specifying the type of Company Class to be returned: Values are:

- 0: (default action) returns all Company Classes
- 1: only Company Classes that are marked as Price Lists
- 2: only Company Classes that are NOT marked as Price Lists

Return Results:

`TDataArray` – An array of `WideString`, contains the available Company Class Names.

Export

```
function GetCompanyStatus(): TDataArray;
```

This method will export a list of all the Company Statuses configured in Khaos Control.

Return Results:

`TDataArray` – An array of `WideString`, contains the available Company Status Names.

Stock Data

Export

```
function GetStockList(DateType: Integer; DateValue: DateTime;
StockItemType: Integer): TStockList;
```

This method expects up to three parameters:

`DateType` – (optional) Determines what date type is used for retrieving the data. Values are:
0: 'REVISION_DATE' - (default) The last time the stock item was edited.
1: 'LAUNCH_DATE' - A date field that can be set against the stock item in Khaos Control by the user. (Can be used for item creation date if this field is filled in when the item is created).
2: 'MOVEMENT_DATE' - The last time this item had a movement recorded against it i.e stock adjusted, stock assigned.

`DateValue` – The date as defined by the DateType parameter. As above, if DateType is not defined, then this will be the date of the last time that the stock item was edited.

`StockItemType` – (optional) An integer value specifying the type of stock items to be returned. Possible values are:

- 0: (default) returns all stock items.
- 1: returns only SCS Parent stock items.
- 2: returns only SCS Child stock items.

Return Results:

`TStockList =`
`property Count: Integer` – Total count of all stock items.
`property List: TStockArray` – An array of `TStockInfo`
`TStockInfo =`

property StockCode: WideString
property OtherRef: WideString – Other reference for the stock code
property StockDesc: WideString – Stock description
property ParentCode: WideString – Parent stock code for the item.
property ParentOtherRef: WideString – Other reference for the parent stock code.
property LastUpdated: TDateTime – The date the stock item was last updated.

Note: The results of this operation should be used in conjunction with the ExportStock or ExportStockCompressed methods described below. Due to customer specific configuration outside of our control, web services may appear to stop responding or timeout during large data transfers. This method allows for a smaller volume of data containing minimal information relating to the stock codes that have changed, so that they can be passed in "chunks" to either ExportStock or ExportStockCompressed.

Export

```
function GetStockListCompressed(DateType: Integer; DateValue: DateTime;
StockItemType: Integer; CompressionMethod: Integer): TCompressedResult;
```

This method expects up to four parameters:

DateType – (optional) Determines what date type is used for retrieving the data. Values are:
0: 'REVISION_DATE' - (default) The last time the stock item was edited.
1: 'LAUNCH_DATE' - A date field that can be set against the stock item in Khaos Control by the user. (Can be used for item creation date if this field is filled in when the item is created).
2: 'MOVEMENT_DATE' - The last time this item had a movement recorded against it i.e. stock adjusted, stock assigned.

DateValue – The date as defined by the DateType parameter. As above, if DateType is not defined, then this will be the date of the last time that the stock item was edited.

StockItemType – (optional) An integer value specifying the type of stock items to be returned. Possible values are:

0: (default) returns all stock items.
1: returns only SCS Parent stock items.
2: returns only SCS Child stock items.

CompressionMethod – An integer value specifying the type of compression to be applied to the XML file returned. Possible values are:

0: Result is compressed and returned within a .ZIP file.

1: Result is compressed and returned as a BZip2 stream.

Return Results:

TCompressedResult =

property Method: String – Either "ZIP" or "BZIP2", depending on CompressionMethod.
property FileName: String – ZIP only. The name of the file containing data stored within the returned ZIP.
property Data: TBinaryData – An array of bytes representing either a standalone ZIP file, or a BZip2 data stream, depending on CompressionMethod.
property OriginalSize: Integer – The size of the uncompressed data.
property CompressedSize: Integer – The size of the compressed data (i.e. length of Data).
property PasswordProtected: Boolean – Reserved. Returns False.

Decompressing the contents of Data results in XML data as defined below. This will match the returned result format of GetStockList above.

TStockList =

property Count: Integer – Total count of all stock items.
property List: TStockArray – An array of TStockInfo

TStockInfo =
property StockCode: WideString
property OtherRef: WideString – Other reference for the stock code
property StockDesc: WideString – Stock description
property ParentCode: WideString – Parent stock code for the item.
property ParentOtherRef: WideString – Other reference for the parent stock code.
property LastUpdated: TDateTime – The date the stock item was last updated.

Note: The results of this operation should be used in conjunction with the ExportStock or ExportStockCompressed methods described below. Due to customer specific configuration outside of our control, web services may appear to stop responding or timeout during large data transfers. This method allows for a smaller volume of data containing minimal information relating to the stock codes that have changed, so that they can be passed in "chunks" to either ExportStock or ExportStockCompressed.

Export

```
function GetStockPotential(StockCode: WideString; MappingType: Integer): WideString;
```

This method expects two parameters:

StockCode – (optional*) If specified the levels for these stock items will be exported out of Khaos Control. The value should be passed as a comma separated string of identifiers. (See Return Results of GetStockList). The identifiers do not have to be 'stock code' – see MappingType next.
* While the Stock Code is technically optional, if it's omitted, an empty result list will be returned.

MappingType – (optional) Determines what "StockCode" will be looked-up against the Khaos Control stock table. Values are:
0: 'STOCK_ID' - (DEFAULT) internal Khaos Control unique reference for the item, not visible to the user.
1: 'STOCK_CODE' - main stock code, which is visible to the user.
2: 'OTHER_REF' - on the stock form as Other Reference (which is sometimes used as the barcode).
4: 'SHORT_DESC' - on the stock form as the main Stock Description

Return Results:

TStockPotential =
property StockCode: WideString – Code to identify the item.
property OtherRef: WideString – Other reference for the stock code.
property StockDesc: WideString – Not used.
property Available: Double – The number of items available for sale.
property OnOrder: Double – The number of items currently on purchase orders waiting to be delivered by the supplier.
property BackOrder: Double – The number of items on sales orders without stock assigned to that item line.
property Potential: Double – The Available + the On Order – the Back Order, i.e. the amount of stock available after all purchase orders have been delivered and back orders have been fulfilled. May be negative.

Export

```
function ExportStock(StockCode: WideString; MappingType: Integer; LastUpdated: DateTime): WideString;
```

This method expects between zero and three parameters:

StockCode – (optional*) If specified these stock items will be exported out of Khaos Control. The value should be passed as a comma separated string of stock codes. (See Return Results of GetStockList). The values do not have to be 'stock code' – see MappingType next.

MappingType – Determines what "StockCode" will be looked-up against the Khaos Control stock table. Values are:

1: 'STOCK_CODE' - main stock code, which is visible to the user (DEFAULT)
2: 'OTHER_REF' - on the stock form as Other Reference (which is sometimes used as the barcode).
4: 'SHORT_DESC' - on the stock form as the main Stock Description

LastUpdated – (optional*) If specified, all stock items that have been updated (changed in Khaos Control) since this chosen date will be exported.

* If both StockCode and LastUpdated parameters are specified the StockCode parameter will take precedence.

Return Results:

XML data as defined in the file KSDXMLStock.xml – details of stock items and their properties.

Note: Also see **GetStockList and GetStockListCompressed** above and the corresponding note.

Export

```
function ExportStockCompressed(StockCode: WideString; MappingType: Integer;
LastUpdated: DateTime; CompressionMethod: Integer): TCompressedResult;
```

This method expects between zero and four parameters:

StockCode – (optional*) If specified these stock items will be exported out of Khaos Control. The value should be passed as a comma separated string of stock codes. (See Return Results of GetStockList). The values do not have to be 'stock code' – see MappingType next.

MappingType – Determines what "StockCode" will be looked-up against the Khaos Control stock table. Values are:

1: 'STOCK_CODE' - main stock code, which is visible to the user (DEFAULT)
2: 'OTHER_REF' - on the stock form as Other Reference (which is sometimes used as the barcode).
4: 'SHORT_DESC' - on the stock form as the main Stock Description

LastUpdated – (optional*) If specified, all stock items that have been updated (changed in Khaos Control) since this chosen date will be exported.

* If both StockCode and LastUpdated parameters are specified the StockCode parameter will take precedence.

CompressionMethod – Determines which Compression Method the Web Service should use when compressing the stock data requested. Available values are:

0: Result is compressed and returned within a .ZIP file.

1: Result is compressed and returned as a BZip2 stream.

Return Results:

TCompressedResult =

property Method: String – Either "ZIP" or "BZIP2", depending on CompressionMethod.
property FileName: String – ZIP only. The name of the file containing data stored within the returned ZIP.
property Data: TBinaryData – An array of bytes representing either a standalone ZIP file, or a BZip2 data stream, depending on CompressionMethod.
property OriginalSize: Integer – The size of the uncompressed data.
property CompressedSize: Integer – The size of the compressed data (i.e. length of

Data).

property PasswordProtected: Boolean – Reserved. Returns False.

Decompressing the contents of Data results in XML data as defined in the file KSDXMLStock.xml – details of stock items and their properties.

Note: Also see **GetStockList and GetStockListCompressed** above and the corresponding note.

Export

```
function ExportStockStatus(StockCode: WideString; MappingType: Integer;
LastUpdated: DateTime; SiteID: Integer; PotentialLeadTime: boolean):
WideString;
```

See **ExportStock** above for a description of the parameters, with the addition of:

SiteID – (optional*) Return results only for specified stock site. If omitted, results for the default stock site will be returned - pass a value of -1 to return results for all stock sites.

PotentialLeadTime – (optional*) Returns calculated Purchase Order Due Date for the stock code in the LEAD_TIME tag, based on the next Purchase Order that has available stock on it at the time of calling the function. Pass a value of -1 to return the calculated date. Omit the parameter, or pass a value of 0 to return the first PO Due Date.

Return Results:

XML data as defined in the file KSDXMLStockStatus.xml – cut-down data of a stock item's key properties.

Note: Also see **GetStockList** above and the corresponding note.

Export

```
function ExportWebCategories(Website: WideString): WideString; stdcall;
```

This method expects one parameter:

Website – The name of the website that you would like the categories returned for. In Khaos Control there is a tree hierarchy that can (optionally) be defined to allow stock items to be contained in various difference categories, for use on the web site. The hierarchy tree can have any number of levels and nesting. A stock item can exist in more than one category. The website parameter is the root of a given tree structure, of which there can be more than one. A stock item in a category can have a difference long and short description which can be unique to that category.

Return Results:

XML data as defined in the file KSDXMLWebCategories.xml – the hierarchy of web categories and the stock items contained within them, including alternate stock item descriptions.

Export

```
function ExportWebCategoriesCompressed(Website: WideString;
CompressionMethod: Integer): WideString; stdcall;
```

This method expects two parameters:

Website – The name of the website that you would like the categories returned for. In Khaos Control there is a tree hierarchy that can (optionally) be defined to allow stock items to be contained in various difference categories, for use on the web site. The hierarchy tree can have any number of levels and nesting. A stock item can exist in more than one category. The website parameter is the root of a given tree structure, of which there can be more than one. A stock item in a category can have a difference

long and short description which can be unique to that category.

CompressionMethod – Determines which Compression Method the Web Service should use when compressing the data requested. Available values are:

0: Result is compressed and returned within a .ZIP file.

1: Result is compressed and returned as a BZip2 stream.

Return Results:

TCompressedResult =

property Method: String – Either "ZIP" or "BZIP2", depending on CompressionMethod.

property FileName: String – ZIP only. The name of the file containing data stored within the returned ZIP.

property Data: TBinaryData – An array of bytes representing either a standalone ZIP file, or a BZip2 data stream, depending on CompressionMethod.

property OriginalSize: Integer – The size of the uncompressed data.

property CompressedSize: Integer – The size of the compressed data (i.e. length of Data).

property PasswordProtected: Boolean – Reserved. Returns False.

Decompressing the contents of Data results in XML data as defined in the file KSDXMLWebCategories.xml – the hierarchy of web categories and the stock items contained within them, including alternate stock item descriptions.

Export

Only available in webservice version 8.144 onwards.

```
function ExportWebCategoriesEx2(WebsiteName: WideString; WebsiteID:
WideString; WebNodeID: WideString; IncludeItems: Boolean): WideString;
stdcall;
```

This method expects between 2 and 3 parameters:

WebsiteName(optional) – The name of the website that you would like the categories returned for. In Khaos Control there is a tree hierarchy that can (optionally) be defined to allow stock items to be contained in various difference categories, for use on the web site. The hierarchy tree can have any number of levels and nesting. A stock item can exist in more than one category. The website parameter is the root of a given tree structure, of which there can be more than one. A stock item in a category can have a difference long and short description which can be unique to that category.

If the WebsiteName is present, then the WebsiteID and WebNodeID are not required, and will be ignored if they are present. If the WebsiteName is not present, the WebsiteID and WebNodeID are required.

WebsiteID(optional) – Used to specify which website to return the information from. Used in conjunction with the WebNodeID.
WebNodeID(optional) – Used to specify a single category within the website specified by the WebsiteID. The WebsiteID and WebNodeID parameters allow returning results for only specific categories, rather than all categories as per using the WebsiteName.

IncludeItems(required) – Determines whether to return stock item data in addition to the category information or only the category information.
Available values are:

0 or False: Only return category information.

-1 or True: Include stock item data and category information.

Return Results:

XML data as defined in the file KSDXMLWebCategoriesEx2.xml - the hierarchy of web categories and the stock items contained within them, including alternate stock item descriptions.

Note: The XML data format differs from the **ExportWebCategories** call. For example, this call does not return SCS child stock codes for SCS parent items; it is expected that this information will be extracted using **ExportStock** or **ExportStockCompressed** on the parent stock item.

Export

```
function GetManufacturer(): TDataArray;
```

This method will export a list of all the Manufacturer Names set up in Khaos Control.

Return Results:

`TDataArray` – An array of `WideString`, contains the available Manufacturer Names.

Export

```
function GetSiteList(): TSiteList;
```

This method will export a list of all the Stock Control Sites configured in Khaos Control.

Return Results:

```
TStockList =
      property List: TSiteArray – An array of TSite
      property CurrentSiteID: Integer – The default Stock Control Site ID
TSite =
      property SiteID: Integer
      property SiteName: WideString
```

Import

```
function ImportStockStatus(StockStatus: WideString; UpdateStockData,
UpdateStockLevels: Boolean): TStockList;
```

This method expects between one and three parameters:

`StockStatus` – KSDXMLStockStatus.xml

`UpdateStockData` – (optional) Indicates whether or not to update stock properties held in Khaos Control, such as the stock items sell price and buy price. Default value is 0 (false). **Warning: Using this function to change stock property values could have serious consequences in the back-office. Please ensure your client is comfortable with the web site updating their back-office stock database.**

`UpdateStockLevels` – (optional) Indicates whether or not to update the stock level held in Khaos Control against the stock item, based on the LEVEL xml tag. It will attempt to use LEVEL as an absolute value, to set a new stock level; it will not adjust the level up or down by a set amount. Default value is 0 (false). **Warning: Using this function to update Khaos Control stock levels should be done only in extreme circumstances, when the web site is the master stock controller. It is NOT required to decrease stock levels when a web order is placed as this will happen automatically when the web order is imported.**

Return Results:

```
TStockList =
```

property Count: Integer – Total count of all stock items updated
property List: TStockArray – An array of TStockInfo for each item updated

TStockInfo =
property StockCode: WideString
property OtherRef: WideString – Other reference for the stock code
property StockDesc: WideString – Stock description
property ParentCode: WideString – The stock code of the stock item's parent (if any)
property ParentOtherRef: WideString – The other reference of the stock item's parent (if any)
property LastUpdated: TDateTime – The date the stock item was last updated

Promotion Data

Export

```
function ExportPriceList(PriceList: WideString): WideString;
```

This method expects one parameter:

PriceList – The name of the price list you would like to export. If no price list is specified then the system will look for a price list that begins with the text "web".

Note: This operation will fail if more than one price list matches the above criterion. This operation will ONLY export Stock Item Price Lists (customer-specific price lists and / or Stock Type price lists of any type will not be returned by the method). If you wish to export other types of Price List, please use the ExportPriceListEx method.

Return Results:
XML data as defined in the file KSDXMLPriceLists.xml

Import

```
function ImportPriceLists(PriceList: WideString): WideString;
```

This method expects one parameter:

PriceLists – XML data as defined in the file KSDXMLPriceLists.xml

Note: This operation will overwrite the price list defined in the above parameter with the XML passed to the method.

Export

```
function ExportPriceListEx(PriceLists: WideString; PriceListType: Integer;
PriceListIDs: Boolean): WideString;
```

This method expects three parameters:

PriceLists – Comma separated list of price list descriptions / company codes to export (see PriceListType parameter).

PriceListType – Determines the type of price lists to be returned. Values are:
1: Return StockItemCustomer price lists as defined in Promotion > Stock Item / Customer. PriceLists parameter should specify a comma separated list of the required companies COMPANY_CODE.
2: Return StockTypeCustomer price lists as defined in Promotion > Stock Type / Customer. PriceLists parameter should specify a comma separated list of the required companies COMPANY_CODE.
3: Return StockItemCustomerClassifications price lists as defined in Promotion > Stock Item /

Price List. PriceLists parameter should specify a comma separated list of the required price lists SHORT_DESC.
4: Return StockTypeClassifications price lists as defined in Promotion > Stock Type / Price List. PriceLists parameter should specify a comma separated list of the required price lists SHORT_DESC.

`PriceListIDs` – Specific Khaos Control ID of price list to be exported.

Return Results:

XML data as defined in the file KSDXMLPriceLists.xml

Export

```
function GetPriceLists(): TDataArray;
```

This method will export a list of all the Price Lists configured in Khaos Control.

Return Results:

`TDataArray` – An array of `WideString`, contains the available Price List Names.

Export

```
function GetKeycodeTypes(): TDataArray;
```

This method will export a list of all the Keycode Types set up in Khaos Control.

Return Results:

`TDataArray` – An array of `WideString`, contains the available Keycode Types.

Export

```
function GetKeycodes(): TKeycodeArray;
```

This method will export a list of all the Keycodes set up in Khaos Control.

Return Results:

`TKeycodeArray` – An array of `TKeycode`

`TKeycode` =

`property KeycodeCode: String` – The code of the keycode

`property KeycodeDesc: String` – The description of the keycode

`property KeycodeTypeDesc: String` – The type of the keycode (see above)

`property SourceCode: Boolean` – Whether or not this keycode can be used as a source keycode, which means "can it be selected and applied manually when a customer is created in Khaos Control".

`property DateCrreated: TDateTime` – The date the keycode was created Note: this property remains misspelt for backwards compatibility.

`property ExpiryDate: TDateTime` – The date the keycode is due to expire. Note: this function will not return expired keycodes.

`property WebUse: Boolean` – If true, return results for keycodes configured for web use

`property MOTOUse: Boolean` – If true, return results for keycodes configured for MOTO use

Export

```
function GetSOPData(Courier, DelRate, CourierBanding, BOGOF, SpecialOffer,
```

```
TelesalePrompt, KeyCode, BarredItems: Boolean): WideString;
```

This method exports the various promotion details/rules available in Khaos Control.

This method expects up to seven parameters, each of which tells the web service to export a different type of promotion. All promotions are exported in the same XML string. The options can be used in combination. See the example XML file for details.

`Courier` – (optional) A list of available Couriers that have been set-up in Khaos Control. Default value is 0 (false).

`DelRate` – (optional) Depending on a destination (country, zone, postcode range) and a courier and/or an order value/weight, rules can be defined to determine the delivery cost of the order. Default value is 0 (false).

`CourierBanding` – (optional) Used in KC to select a courier against an order when the order is being created, based on the order's weight. Default value is 0 (false).

`BOGOF` – (optional) Buy One Get One Free – the Trigger item determines what Offer items are available. Default value is 0 (false).

`SpecialOffer` – (optional) A simple list of stock items that are on Special Offer. Default value is 0 (false).

`TelesalePrompt` – (optional) Telesale prompts and rules can be used to "trigger" certain behaviour, such as to display a message or to trigger a Keycode (see next) via the "apply keycode" tag, based on certain conditions. Default value is 0 (false).

`KeyCode` – (optional) A keycode is a campaign code and the most widely used of the promotions. It can be used to offer discounts, etc against orders. Using a keycode it is possible to define when an order should be discounted or the delivery amount adjusted. It is also possible to define certain stock items as being discounted when using a specific keycode. Default value is 0 (false).

`BarredItems` – (optional) Products that have been barred against customers, either specifically, or by Company Class. Default value is 0 (false).

Return Results:

XML data as defined in the file KSDXMLSOPData.xml

Other Data

Export

```
function GetVersion: WideString;
```

Return Results:

The version number of the kOS objects used to compile the web service. This can be used to compare against the back office Khaos Control application version number.

Export

```
function GetAgentList(): TDataArray;
```

This method will export a list of all the Agents configured in Khaos Control.

Return Results:

`TDataArray` – An array of `WideString`, contains the available Agent Names.

Khaos Control – Notes

Required Information to Start Integration

Some methods / XML feeds in the web service will require you to pass parameters that can be variable in Khaos Control. With this in mind, it would be advisable to ask your client (Khaos Control User) for the following information:

1. **`Website Name`** (required) – This is used in **`ExportWebCategories`** and is the top level node in the Web Configuration Category Tree screen in Khaos Control.
2. **`Price List Name`** (optional) – This is dependent on your client using price lists and you using them on the website.
3. **`Company Class for Orders`** (recommended) – When customers are created on an order import, what Company Class should they be created with?
4. **`Sales Source for Orders`** (recommended) – This can be set up in Khaos Control and is passed down in the KSDXMLOrderImport.xml file.

Matching Customer Records

When importing customer records either via the **`ImportCompany`** or **`ImportOrders`** methods, the client's Khaos Control/Web Service will be initially configured to try and match the incoming customer record to one held already in the database. The default matching hierarchy is as follows:

Import Company

Initially **`ImportCompany`** will try to match on the following fields (when they are populated in the KSDXMLCustomer.XML file):

1. Company ID
2. Company Code
3. Other Ref (recommended)
4. Web User

If these fields have been omitted from the XML file, have been left blank, or are not matched with existing Customer records in Khaos Control, then the system will work through each of the following fields in the XML file:

1. Invoice Contact Surname
2. Invoice Address Postcode
3. Invoice Address Town
4. Invoice Address Line 1
5. Invoice Contact Forename
6. Currency

Initially all these fields will be matched on, then the system will remove one at a time (in the order they are listed here) until a match is found, however, the system won't remove matching types that have been set up to match on by default. i.e.; the System default is to match on Surname and Postcode so that when the system loops through this list, it will never remove Surname or Postcode.

Note: It is possible to change the default matching criteria for each instance of Khaos Control, if this is required, please contact Khaos Control Solutions.

Note: We recommend that the Other Ref field be used as the unique ID for the customer on the website. This will allow you to match up your records with the ones held in Khaos by matching on this field.

Note: We do not recommend assigning more than one invoice address per company record as this can cause complications within other areas of the system. Khaos Control will automatically use a customer's Invoice Address as the Delivery Address, therefore, where these are the same for a web customer then only an Invoice Address should be specified in the XML file. This will avoid unnecessary duplication of address and / or contact data.

Note: Whether a Contact is related to the Invoice Address or Delivery Address depends on the address type of the address that the Contact is associated with.

Note: When importing customers into Khaos Control, the XML file allows you to pass through ID's. These should only be used in specific circumstances and when discussed with Khaos Control Solutions.

Import Orders

Initially `ImportOrders` will try to match on the following fields (when they are populated in the KSDXMLImportFormat.XML file):

1. Company Code
2. Other Ref (recommended)
3. Web User

If these fields have been omitted from the XML file, have been left blank, or are not matched with existing Customer records in Khaos Control, then the web service will check the following KhaosControl.ini file option:

```
[OrderImport]
MatchCodeOnID
```

When this is set to '-1', the system will attempt to match the Company Code in the XML file against a customer in the Khaos Control database by Company ID.

If the customer is still not matched, the system will work through each of the following fields in the XML file:

1. Surname
2. Postcode
3. Town
4. Address
5. Company Name
6. Forename
7. Currency

Initially all these fields will be matched on, then the system will remove one at a time (in the order they are listed here) until a match is found, however, the system won't remove matching types that have been set up to match on by default. i.e.; the System default is to match on Surname and Postcode so when the system loops through this list, it will never remove Surname or Postcode.

Note: It is possible to change the default matching criteria for each instance of Khaos Control, if this is required, please contact Khaos Control Solutions.

VAT

Import

Khaos will calculate VAT on a line by line basis. Please see the example below for a guide on how to calculate the VAT per item line.

(Round(Net – DC%) * Qty) * (VAT Rate / 100) = Line Gross Total.

Note: If you calculate VAT differently on the website then there could be differences in the amount you have charged the customer and the Order Total in Khaos Control. Whether the calculations for the customer are done in Gross or Net depends on the Company Class within Khaos Control and cannot be set using the web services. Please refer to "Khaos Control Sales Order Calc v1.0.xls" for detailed calculation methods used by the web service.

Dealing with Discounts and Keycodes

Import

Within Khaos Control you have the ability to attach a Keycode to the Sales Order that can apply a discount however; this doesn't work in the same way when using the web service.

When an order is imported into Khaos Control via the web service, all prices passed down are locked. The reason for this is that the payment has generally been taken and the prices the customer has paid would therefore need to stay the same. On that note, when attaching a Keycode to an order imported via the web service this will only be used as a reference and will not apply any discounts; it is up to the web developers to apply discounts to the items before they are imported.

Note: If you need any more information about Keycodes. Please contact your Khaos Control Client.

Displaying Stock Items on the Website

Import

There are a number of ways in which your Khaos Control Client may have configured their stock items in Khaos Control with regards to flagging whether or not they should be displayed on the website. As a result, there are four main elements that you will need to take into account when deciding whether or not to display an item on the website:

1. The DELETED Stock XML feed tags denotes whether or not an item should be displayed on the website. As described in the example XML document for the Stock feed, the DELETED tag is a calculated value based on the DISCONTINUED and WEB values. Any item that has a DELETED value of '-1' should not be displayed for sale online.

2. The RUN_TO_ZERO Stock XML tag can also impact whether or not a stock item should be displayed for sale online. Any item that has a RUN_TO_ZERO value of '-1' AND a LEVEL of <= 0 in the Stock Status XML feed should not be available for sale on the website. Furthermore, any sale of a this kind of item must not take the item's level below 0. Allowing orders to be placed for an item in this state will result in the order failing to import into Khaos Control.

3. The Stock Status XML feed provides two key pieces of information that could impact this further:
    a. STATUS
    b. LEVEL

4. Finally, the Web Categories XML feed will provide you with detail on where, and potentially whether, an item should be displayed for a specific website.

Note: It is important to discuss how to approach the above with your Khaos Control Client directly, as each client's requirements will be different, and could even change from website to website, depending on the target market and the items being sold.

sales@khaoscontrol.com
www.khaoscontrol.com
0845 257 5111

Contacting the Web Service

Important Note: This element of the integration is not the responsibility of KCSL. The notes here are for guidance and help purposes only and are not meant to be exhaustive. If you are have trouble generating an interface to our web service there are many books, internet articles and training courses available on the subject.

To connect to the web service, you will have to generate an "interface". The way in which is done varies from one development environment to the next. There is usually a wizard available to "Import a WSDL file", or "Generate WSDL interface file", or "Add Web Service", or "Add Interface".

Generic Test Web Service URL:

http://office.keystonesupport.net:8084/DemoSoap/KSDSoap/KhaosKSD.exe

WSDL URL:

http://office.keystonesupport.net:8084/DemoSoap/KSDSoap/KhaosKSD.exe/wsdl/IKosWeb

General Khaos Control Web Service Notes

Associated XML Files

Please review the sample files that are included in the integration zip file, as they contain additional comments and help that will aid understanding of what is required.

Testing Your Files

Common problems we have seen in the past that should be considered when producing the files:

- Use a appropriate XML file parser – tools to process XML should come with almost every programming language. Trying to process an XML file "manually" will normally cause problems as we can (and do) add extra fields to the files as newer versions of Khaos Control add more options & capabilities. If your website software is using a proper XML parser then it should be able to automatically ignore any fields it doesn't require. Likewise if Microsoft change the XML standards then the parser will automatically handle this, whereas a manually accessed file will stop working because the XML is no longer in the same format as before.
- Be sure to test what happens if a user on the website enters data containing special symbols, particularly < > & £ . Again, this should be handled automatically provided you are using an XML parser, but this is definitely worth checking!
- Be aware that most of the fields in the files are intended to be read by the system and are not meant to be "human readable". For example, the fields in the order files containing payment/item amounts should be simple amounts such as '12.99', they should not contain any formatting, e.g. £1,200.99, as this will prevent Khaos Control reading the information correctly. Of course fields which contain text (such as order notes, etc) can have any data within them.
- Try to limit size of string values passed to the web service. This is not usually picked up in testing, but as soon as users are let loose on the web site they will usually type something you are not expecting, like the whole address into the first line – we've seen it happen! This fails to import as the string is too long.
- Khaos Control uses the Microsoft XML libraries to read and write XML files. This is the same code that Internet Explorer uses; so, as a rule, if Internet Explorer refuses to load in an XML file because it is not valid, then Khaos Control will also be unable to read it.
- Dates should always be specified using the format: `YYYY-MM-DD` or `YYYY-MM-DDTHH:MM:SS` (e.g. `2010-05-01` or `2010-05-01T11:56:02`).
- Times sent to Khaos Control are not intentionally rounded, however, this will depend on the level of precision that you're using. The Khaos Control database will store time to milliseconds, rounded to 3dp.
- KSD always recommend that when using times to pull records that have changed since a certain time, web developers include a buffer, to ensure that all updates are captured. This will ensure that any client and / or server time configuration issues that the customer may have do not impact the integration.

Web Service Configuration

Each individual web service has a Configuration file that allows KSD to set and maintain default behaviour specific to the customer's instance. Settings that can be controlled via the Config file include:

- Database Name
- Database Server

- Khaos Control User for the Web Service
- Default directories
- Default logging settings
- Convert strings within XML files into [CDATA].

If you have any questions about, or require any of these settings to be amended, please contact KSD by emailing support@keystonesoftware.co.uk or by calling 0845 25 75 111 (Option 1).

Archiving Imported and Exported Files

Each individual web service method will, by default, archive the XML files that have been sent via it. This data is useful when testing a new integration, or an amendment to an existing integration and is also useful for the KSD Support team if any issues arise with regards to the web service in the customer's Live environment. However, this data is of limited use in the long term and can, over time, clog up your web service server unnecessarily.

If you have any questions about deleting archived XML, please contact KSD by emailing support@keystonesoftware.co.uk or by calling 0845 25 75 111 (Option 1).

File Specific Notes

Order Import File (KSDXMLImportFormat.XML)

a) Comments

- Please do not replicate the comments as part of the integration, as they just clutter up the file and make it unnecessarily large. The comments are just there to help you understand the tags.
- When specifying a stock code for a Pack Header item, Khaos Control will automatically add the pack header's child items onto the sales order when it is imported into Khaos Control.

b) Required Fields

- Company Code should be specified if the website knows an existing customers' reference. The website should **not** generate codes for new customers unless it can be sure of generating a code which will not 'clash' with one in Khaos Control.
- The IS_NEW_CUSTOMER field must be specified.
  If set to zero (false) then Khaos Control will attempt to locate a matching customer in the database using the data you have provided (see Matching Customer Records on page 3).
  If set to -1 (true) then Khaos Control will **always** create a new customer: this is not normally the desired behaviour, so we would normally recommend setting the field to zero.
- Invoice Address **must** be specified; at a minimum, address line 1, town, postcode, and surname.
- Delivery Address is optional, if not specified the invoice address is used for delivery.
- Associated Ref is compulsory: this **must** contain a unique reference for the order. This is used to make sure the same order does not get imported twice.
- For fields such as Keycode and Sales Source, if specified, the keycode/sales source/etc must already exist in Khaos Control. Note these are user-configurable fields, so you should consult whoever is setting up your Khaos Control system to see what values are valid. Other fields like this include:
  - Brand
  - Company Class
  - Company Type
  - Sales Source
  - Free Item Reason

- For order items, the following fields must be specified: Mapping Type, Stock Code, Order Qty, and one of either Price Net **or** Price Grs. An order must contain at least one order line. Description is not required; if omitted, the default description against the stock item in Khaos Control is used for that order line.
- If an order item line is using anything other than the standard tax rate, you must use the Price Net and Tax Rate flags.
- The Mapping Type field should normally be set to "1" to instruct the import routine to find the stock item by using stock code.
- The Option Ref fields are **not** required unless the information specified in stock code is insufficient to locate an exact item. For example, if your stock codes are not unique then the Option Ref fields are used to drill down to the specific item needed.

c) Maximum Field Lengths

- Company Code – 10 characters
- Company Name – 50 characters
- Address Fields – 35 characters per field, except for postcode (10 characters), telephone numbers (19 characters), and email (50 characters)
- Associated Ref – 15 characters
- PO Number – 30 characters
- Notes fields – 2000 characters
- Item description – 250 characters
- Extended description – 150 characters per line; however, any number of lines can be embedded using a CDATA block.

Customer Import File (KSDXMLCustomer.XML) and Catalogue Request File (KSDXMLCatRequest.XML)

a) Required Fields

- Essentially the same as the order import format. Company Code should be specified if the website knows an existing customers' reference. The website should **not** generate codes for new customers unless it can be sure of generating a code which will not 'clash' with one in Khaos Control.
- Again pay particular attention to the IS_NEW_CUSTOMER field.
- At least one address **must** be specified; at a minimum, address line 1, town, postcode, and surname.

b) Field Lengths

- Same as the customer fields from the order import format

Contact Address

Technical questions regarding integration should be directed to
development@keystonesoftware.co.uk

sales@khaoscontrol.com
www.khaoscontrol.com
0845 257 5111

Changes and Updates List

The revision history for this document can be found in the accompanying document RevisionHistory.pdf