SellerDeck Desktop 2016 Extensions

Author:	Hugh Gibson
Date:	22 September 2016
Revision:	1.0.4
Document:	SellerDeck 2016 Extensions.docx

Revision History

Revision	Date	Author	Comments
1.0.0	12 June 2015	Hugh Gibson	Created from Design documents
1.0.1	7 October 2015	Hugh Gibson	Modifications for release
1.0.2	15 December 2015	Hugh Gibson	Corrections for 16.0.1 release
1.0.3	6 January 2016	Hugh Gibson	Added Development licence keys
1.0.4	22 September 2016	Hugh Gibson	Revised for 16.0.2 release

Review History

Reviewed version	Date	Reviewer	Comments
1.0.0	15 June 2015	Jawahar Jeyaraman	Reviewed and updated the API request/response XML

Contents

1 Introduction

The SellerDeck Desktop application allows additional functionality to be added via layouts and external programs. A number of Plugins and Add-ons have been created that work in this way. However an area that has been missing is the ability to coordinate updates to the database with other users of the application.

This has been remedied in SellerDeck 2016 by adding an API. Access to the API is tightly controlled and is only allowed from a licensed ***Extension***.

Extensions are installed separately to SellerDeck Desktop. If an Extension is to be used on a PC it has to be installed and licenced on that PC.

Extensions don't function as separate programs, but present their user interface within the SellerDeck Desktop application.

Extensions can add functionality in a number of areas, but the key facilities provided are:

- Tabs in the main user interface
- Menu items and dialogs
- Synchronisation with other users and cache updates via the API
- Licensing

This document gives details of the architecture used to implement Extensions as well as the interfaces provided.

In SellerDeck 2016 an initial Extension has been developed to support eBay order management. The facilities provided within SellerDeck Desktop have initially been focussed on enabling this Extension to operate. Other facilities will be added in the future as the need is identified.

An Extension Development Kit gives information on developing Extensions in PHP building on the library of functionality created by SellerDeck Ltd.

For general discussion about Extensions see the forums at http://community.sellerdeck.com/forumdisplay.php?f=161.

To contact SellerDeck Ltd regarding a new Extension, use the email address sdd-extensions@sellerdeck.com.

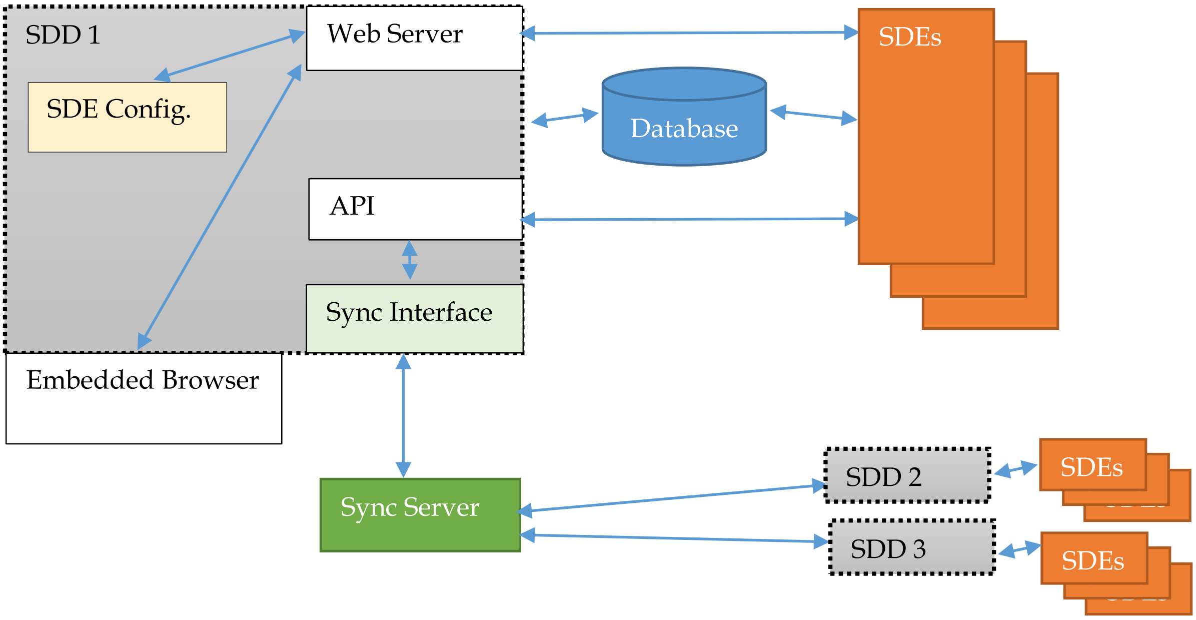
Note:

In this document SDD refers to SellerDeck Desktop – the Windows application - and SDE refers to SellerDeck Extension.

2 Architecture

2.1 Diagram

The architecture to support SDEs is:



2.2 Internal Web Server

SellerDeck 2016 has added a Web Server to support Extensions. The Web Server is internal to the C++ application and is configured to operate Extensions efficiently.

An Extension runs as a CGI application, and serves up web pages. These are shown within SDD using an embedded browser and are used to interact with the Extension.

The web pages cannot be accessed from outside SDD.

In addition, there are pre-defined HTTP requests made to the Extension to verify licencing, configure use of the API and so on.

Initially support is provided for Extensions written in PHP, or compiled executable programs. Other interpreters may be added as required.

2.3 API

SellerDeck 2016 adds an API which can be called by an Extension. The API is predominantly used to tap into the in-built synchronisation process in SDD. This is used to lock resources for editing (e.g. an Order) and to notify other instances of SDD when a resource has changed so that caches can be updated.

Access to the API is controlled so that only Extensions can use it.

2.4 Database

There are no facilities in the API for accessing the database. The Extension is responsible for accessing the database directly, and must coordinate all database activity with SDD using resource locking provided by the API.

The Extension must be able to use Access databases as well as SQL Server databases.

3 Technical Details

3.1 SDE Structure

The key technical requirements of a SDE is that it is able to be integrated into a Web Server using the Common Gateway Interface – CGI see http://en.wikipedia.org/wiki/Common_Gateway_Interface. This runs an executable to give the result of a HTTP request. Environment variables are set up with information about the request, and STDOUT is taken and sent back to the browser.

Interpreted languages such as PHP, Python and Perl provide this facility as standard. Compiled executable programs can also work in this environment.

3.2 Standard Interpreters

SellerDeck Desktop installation includes standard interpreters.

Initially the most recent Windows build of PHP is installed. Currently this is PHP 5.6 – see http://windows.php.net/download#php-5.6 . This also includes the Visual C++ Redistributable for VS 2012. PHP is installed to the SellerDeck installation folder.

The internal Web Server knows about the installed interpreters and doesn't have to be told where they are using a shebang (`#!`) line in the script files.

Perl may be installed in future, though this will be for running the shop scripts locally e.g. to pre-process filter pages.

Note that the internal PHP engine used when rendering layouts is separate to the Web Server PHP engine.

3.3 SDE Installation

Extensions are installed separately to SDD, on every PC where the Extension is to be used. They appear in the Programs and Features list in the Control Panel, and have a name beginning "<extension name> <version>" with Publisher set appropriately.

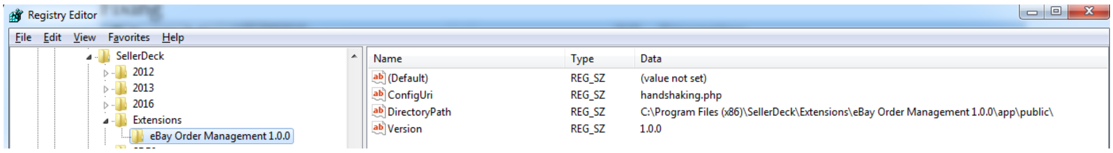
A registry entry indicates the presence of the Extension to SDD. This includes basic information on how to call the Extension.

The registry key is

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Sellerdeck\Extensions\<extens
ion name> <version>
```

The key information for the Extension is

- Extension Name – constructed using ANSI alphanumeric characters and spaces
- Install Folder of the Extension (DirectoryPath)
- Relative location (from the Install Folder) of file which responds to an initial configuration query in the Web Server (ConfigURI)
- [optional] URI pattern – any incoming URIs that matches this pattern will be redirected to the Extensions installation folder (URIPattern)
  Usually its value is like /extension/%s/ where %s will be replaced by Extension Name
- [optional] Icon path – the path of an icon to display on the Extension's tab. The accepted resolution is 16x16 (IconPath)



Name	Type	Data
(Default)	REG_SZ	(value not set)
ConfigUri	REG_SZ	handshaking.php
DirectoryPath	REG_SZ	C:\Program Files (x86)\SellerDeck\Extensions\eBay Order Management 1.0.0\app\public\
Version	REG_SZ	1.0.0

On start-up, SDD makes a standard request to the ConfigUri asking for more detailed configuration information. The XML response indicates licensing, capabilities and the interface requirements of the Extension. See Appendix A below for more information on this exchange.

3.3.1 Versions

Different versions of an Extension are installed separately. The Extension version consists of 3 numbers, following the Semantic Versioning 2.0.0 standard - http://semver.org/. If two versions of an Extension are installed, the most recent version is shown in the user interface in SDD.

In addition each Extension includes information on which version of SDD it must operate with. For example, if an Extension uses facilities only provided in 16.0.1 then that information is included in the initial Handshaking process. If the Extension is installed with 16.0.0 then it won't operate.

3.4 Licensing

Development of an Extension requires an Extension Licence Key Base – EKB - to be allocated for the Extension. Licence Keys can then be created from the EKB by SellerDeck. A Licence Key will authorise operation of an Extension on a single computer.

SellerDeck will administer allocation of EKBs; and creation, sale and registering of Licence Keys.

Extensions will be tied to a single PC. If any Extension is required to run on multiple PCs, it will have to be installed separately on each and will require multiple licences.

More details are given below.

3.4.1 Register of SDEs

SellerDeck maintain a register of SellerDeck Extensions. SDE Developers are required to register each Extension. This may have some cost associated with it for administration of Extension licensing, testing, and creating an installer.

The register contains:

- the name of the Publisher,
- the URL for the Extension (which should be specific to that Extension),
- Extension Prefix,
- evaluation terms i.e. None, 7, 14 or 28 days evaluation,
- annual or perpetual licence
- and uniqueness of licence keys across a network.

The URL is given in the Extensions Housekeeping page and is used to give background information about the Extension as well as how support is obtained for the extension.

On registering, an Extension Key Base (EKB) will be allocated. The SDE Developer will be given an ETOKEN which contains the hashed version of the EKB. The ETOKEN is required during Extension configuration in the SDD environment.

The name given when registering the Extension is used in the calculation of the ETOKEN, so the same ETOKEN can't be used for different Extensions.

3.4.2 Development EKBs

When initially developing ideas for Extensions a private EKB can be requested. This **must** only be used for private Extension development as the same EKB may be allocated to different Extension developers. When development has been completed, a normal EKB can be given with full details.

10 private development EKBs are available, from EKB 10 through EKB 19. When developing Extensions using these EKBs a test licence key can be used to licence the Extension in your local installation of SellerDeck Desktop. These keys **must** not be given to customers.

The licence keys are:

EKB	Licence Key 1	Licence Key 2
10	`qyc-har-efy-4Bj`	`h5h-d3a-Jdp-5a9`
11	`zTh-vED-AbK-DyE`	`BRk-Jkw-q2K-eCG`
12	`55T-hhk-K10-TAU`	`hxF-44t-gC5-j2D`
13	`Ezt-qzu-REq-zpe`	`pwc-rvy-vzD-k8R`
14	`24e-Pha-ugK-105`	`gFJ-AM4-B3e-y3z`
15	`hSw-w7g-a4g-CN0`	`jjr-uTs-6en-5aU`
16	`neP-zBy-0ad-zJB`	`pD8-3Fe-BUs-UTx`
17	`pyF-qjq-gCH-9JU`	`pe4-9d7-E51-rGF`
18	`7b2-nv9-BUU-aNr`	`k9M-CfG-ndT-wuE`
19	`Cws-mJG-0fe-2gH`	`Apx-wn6-vfs-By7`

3.4.3 Licence Key Management

SellerDeck provides facilities for generation and invoicing of Extension licence keys. Payment will be forwarded to the SDE Developer (minus an administration fee) under the Partner programme.

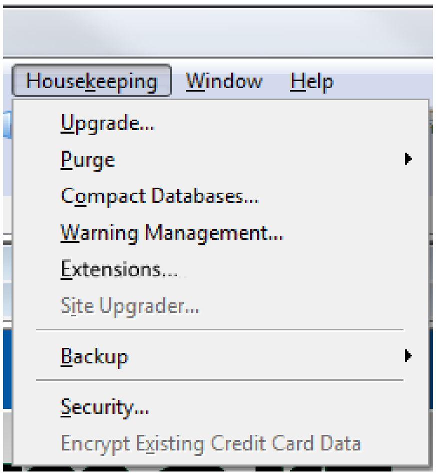
3.4.4 Licence Key User Interface

A list of installed Extensions is provided within SDD. A licence key can be entered against each Extension. An indication will be made about whether the licence key is valid or not.

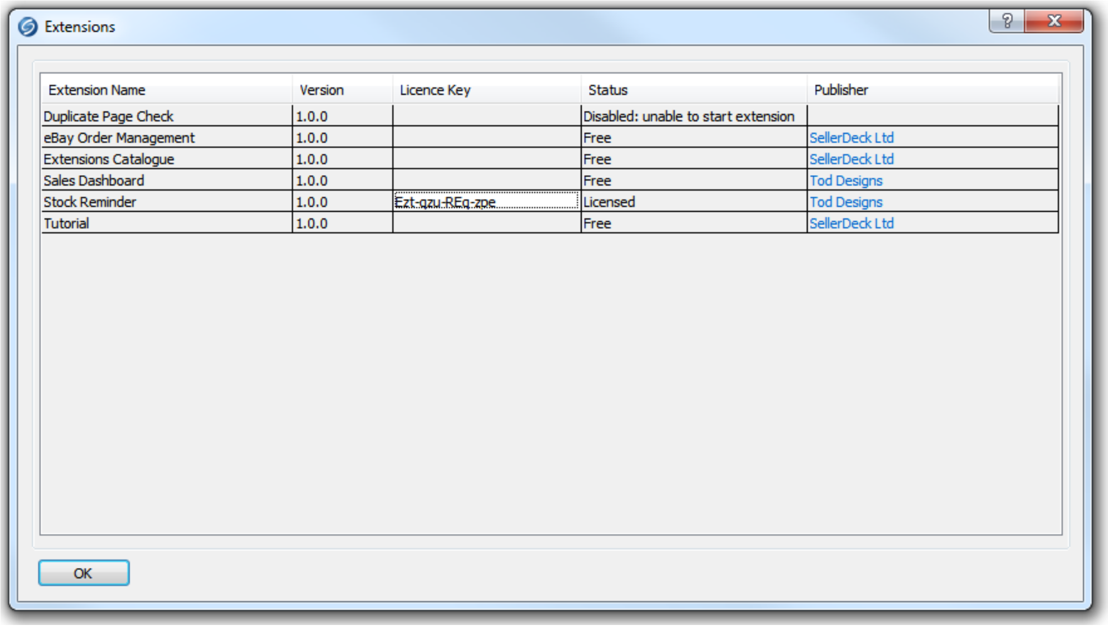
If the licence key is not valid then the Extension will be ignored and will not operate.

A licence key will be required for each computer running the Extension, unless the Extension is marked as Free. Extension licensing can be configured so that the same licence key can be used on all computers on a network, or requiring unique licence keys on all computers on a network.

The licensing dialog is called from a new ‘Extensions’ option in the ‘Housekeeping’ menu. This option is visible if at least one Extension is installed:



Selecting ‘Extensions’ opens a dialog wherein the installed Extensions can be viewed and licensed.



Extension Name	Version	Licence Key	Status	Publisher
Duplicate Page Check	1.0.0		Disabled: unable to start extension	
eBay Order Management	1.0.0		Free	SellerDeck Ltd
Extensions Catalogue	1.0.0		Free	SellerDeck Ltd
Sales Dashboard	1.0.0		Free	Tod Designs
Stock Reminder	1.0.0	Ezt-qzu-REq-zpe	Licensed	Tod Designs
Tutorial	1.0.0		Free	SellerDeck Ltd

The dialog displays a grid showing:

- Extension Name – showing the name of each installed Extension.
- Version – the version number for the Extension.
- Licence Key – where a licence key for the Extension can be entered. If the key entered is not valid for the Extension, it will be rejected with a warning, in the same way as a licence key for the main SellerDeck application.
- Status – will show ‘Licensed’ if a valid key has been entered, ‘Evaluation: xx days remaining’ during an evaluation period, and ‘Unlicensed’ otherwise.
- Publisher – show the publisher of the Extension. Clicking upon the link opens the publisher URL in a web browser.

Deleting the licence key will have the effect of unlicensing the Extension and will not trigger a new 28/14/7 day evaluation period. On a multi-user system it will free up the key for re-use on another PC.

3.4.5 Evaluation Period

It is possible to evaluate SDEs.

When first installed, there may be an evaluation period for the Extension. This may be 28 days, 14 days, 7 days or no evaluation period. The length of the evaluation period is encoded into the ETOKEN. After this period has expired all operations of the Extension will be disabled.

Extensions that carry out a one-off operation e.g. to reconfigure the database in some way should have no evaluation period.

3.4.6 Annual or Perpetual Licence

The Extension may be licensed in two ways. A Perpetual Licence Extension only requires a single licence key, and will never expire. An Annually Licenced Extension requires a new licence key each year to continue operating. A warning message is given when starting SDD as the expiry date approaches. Adding a new licence key will add 365 days to the licence period remaining.

3.5 Data

Extensions must not create or modify any file under the SellerDeck installation folder. They should also not store information in the Extension installation folder.

Persistence of information should be considered. For example data held in the site database will be preserved during a snapshot. That might not be appropriate for user settings, but would be relevant for external API licence keys.

3.5.1 Extension Prefix

All entities stored outside the Extension itself must have a consistent prefix. This is allocated when registering for an Extension. It uniquely identifies files, layouts, tables or other entities. It will prevent conflicts with other Extensions e.g. for files stored in the site folder; or database tables. When 10 Extensions are installed it will enable users to easily find relevant

The prefix take the form `sde_<extensionname>` e.g. `sde_StockReport`

When used with files or folders the prefix must be followed by an underscore e.g. `sde_StockReport_`.

Some files may be stored in folders identified with the Extension e.g. Log files. These do not have to include the Extension Prefix.

3.5.2 Accessing the Site Database

SDEs are allowed to directly access the Site Database. The SDE will be required to work with SQL Server and Access databases. In the future there may be a database API within SDD so it is recommended that a clean MVC architecture is used in the Extension to support this change.

There is a database schema definition in the Extension Developers Kit for the SellerDeck Desktop application. This replaces the older FilesAndTables.doc and is used for database validation among other things.

Care must be taken when accessing the Site Database particularly when working with common tables that are cached in SellerDeck. The API must be used to lock access to the resource e.g. Order, Product prior to any changes.

3.5.3 Extension Tables in the Site Database

Additional tables can be added to the Site Database. These must follow a strict naming system so that the data can be managed properly. No other changes to the structure of the database are allowed. No additional SetupInfo or similar records are allowed.

The database tables must have names that begin with the Extension Prefix. It is up to each Extension how it handles versioning etc. It may use separate tables for each version, or use the same tables and dynamically check for schema changes etc. Extensions must avoid leaving temporary or superseded tables in the database.

If, for example, additional data is required to be stored for a product for the Extension, then a separate table with the same primary key as the Products table must be created. This guarantees that the functionality within SellerDeck for importing and managing data is not impacted.

Setup information for the SDE must be stored in a database table created by the SDE for that purpose.

Coordination of database access with different instances of the SDE on other machines can be managed using the Extension lock/unlock group in the SDD API. This allows coordinated access to a set of private set of resources. For more information see Appendix B below.

All SDE tables will be included in full snapshots.

3.5.4 Site Folder

Information that is relevant for the site, e.g. generated images, can be stored in the site folder. They can be stored in a folder structure using the Extension Prefix:

```
Site
      sde_StockReport
```

Files can also be stored in the Site folder directly but must include the Extension Prefix.

Note that files for upload must include the Extension Prefix so there are no conflicts when the files are uploaded to the website. All website files are stored in a single folder.

Only files that are referenced in the upload will be included in snapshots.

3.5.5 ProgramData

Information that is relevant for the current machine should be stored in the ProgramData folder. This might be local personalisation information for the Extension, temporary files or similar.

ProgramData also contains a Log folder (introduced in SellerDeck 2016) which will have a SDE section for individual SDE logs. For example
`C:\ProgramData\SellerDeck\SellerDeck 2016\Logs\Sites\Site1\sde\Design Backup 1.0.0`

The paths for storing this information are transferred in the initial configuration of the Extension as the merchant may have chosen a non-standard location. See Appendix A for more information.

The Extension must assume that the folders associated with the paths haven't been created and should create them as necessary.

3.6 Interaction with the Extension

3.6.1 Configuration HTTP Requests

Configuration requests originate internally to SDD. Internally these use the HTTP protocol rather than any more direct route e.g. reading configuration files directly from the disk. A standard set of requests and expected responses are defined. In general XML is used for transferring information.

Typical information that is requested is SDE version, and a list of compatible versions of SDD.

3.6.2 User Interface HTTP Requests

User Interface requests are also be made via HTTP but this is via a browser. Browser instances are embedded in SellerDeck. They are given a URL that is served by the internal web server, and which refers to a particular SDE.

Typical information that is given via UI requests could include a Help page. This is shown as a browser instance embedded in a dialog, and is available from a list of installed SDEs shown in a menu in SDD.

The SDE may also request (in a response to a Configuration request) a separate tab within SDD, and show HTML within the embedded browser in the tab – similar to the existing Services tab. However, the SDE can use JavaScript and forms etc to create a web application.

All URLs, both configuration and UI, should be relative to the URI apart from `/site` and `/root` – see below.

All HTML pages must have the meta tag below in the header:

```
<meta http-equiv="X-UA-Compatible" content="IE=10">
```

This forces the embedded instance of Internet Explorer to work with newer HTML standards. Nevertheless, all HTML must be tested carefully within SDD to ensure it looks OK. There may be differences with other browsers used for testing (using the `–extdebug` command line flag).

3.7 SellerDeck API

The SellerDeck API is configured as a SOAP service. The Extension is told the port number and the API key during the initial configuration phase.

Calls to the API are used to request a lock on a given resource – e.g. an order with a specific order number; or a product with a given product reference. If the lock is granted, the Extension is able to update the resource in the database. While locked, the resource may be shown as locked in the SDD UI. When the lock is released all instances of SDD are notified so they can update their cache of the record, and display the resource as available for editing.

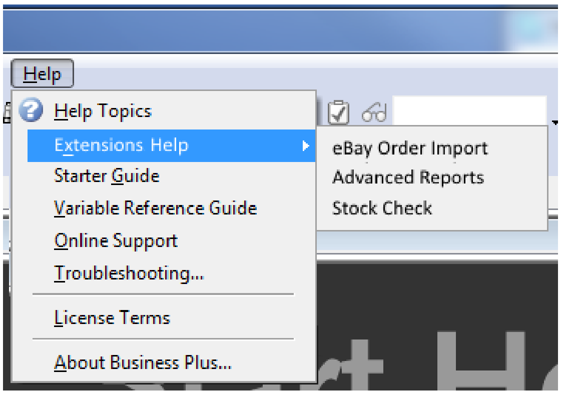
In addition to locking standard resources, Extensions can lock their own private resources. This can be used to coordinate access to the Extension’s database tables.

More details are given in Appendix B “SellerDeck API” below.

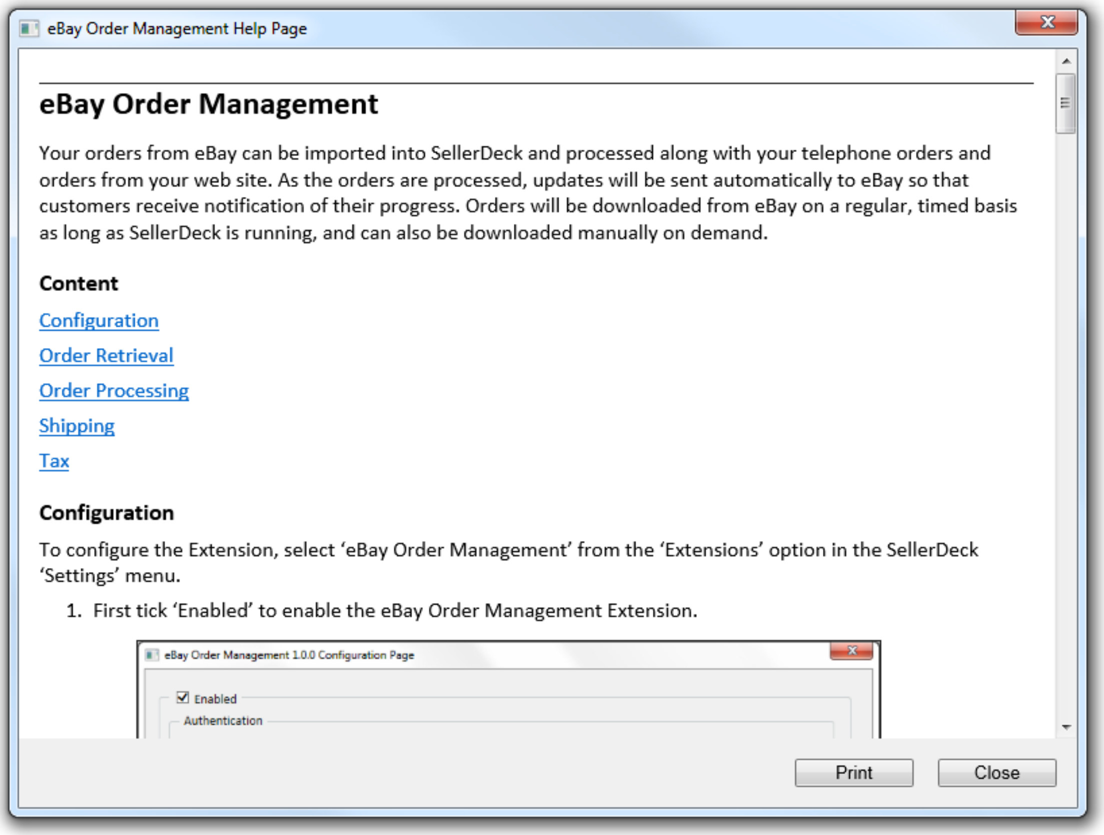
3.8 User interface

3.8.1 Help

Extensions must include a help HTML page, which is accessed via a new ‘Extensions’ option in the ‘Help’ menu. This option is visible if at least one Extension is installed. It pops out a submenu, with each Extension listed:



The Help pages should include a Print button. For an example of how to create this see the tutorial source code included in the Extension Development Kit.



eBay Order Management

Your orders from eBay can be imported into SellerDeck and processed along with your telephone orders and orders from your web site. As the orders are processed, updates will be sent automatically to eBay so that customers receive notification of their progress. Orders will be downloaded from eBay on a regular, timed basis as long as SellerDeck is running, and can also be downloaded manually on demand.

Content

Configuration

Order Retrieval

Order Processing

Shipping

Tax

Configuration

To configure the Extension, select ‘eBay Order Management’ from the ‘Extensions’ option in the SellerDeck ‘Settings’ menu.

1. First tick ‘Enabled’ to enable the eBay Order Management Extension.

Help could get rather long for a single page. Strategies for managing this include

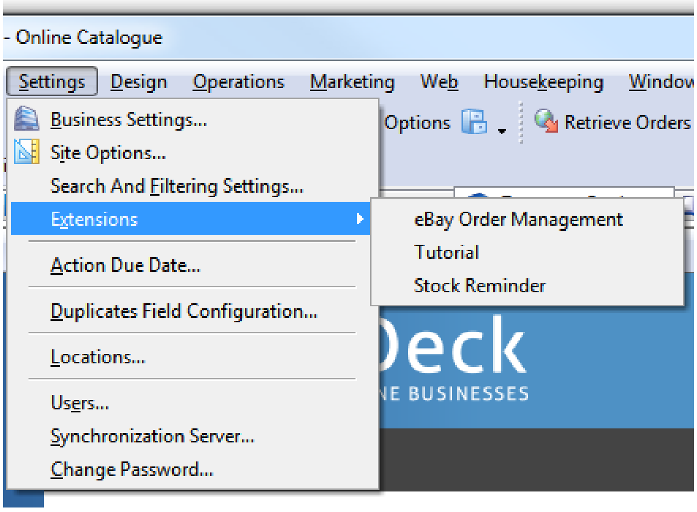
- Table of contents
- Back to top links
- Fixed navigation elements e.g. top bar (which should be hidden when printing). A limited number of buttons can be placed in the bottom bar in the dialog.
- “Find” within the page driven by JavaScript
- Multiple help pages, with links from the navigation elements or table of contents.

3.8.2 Configuration

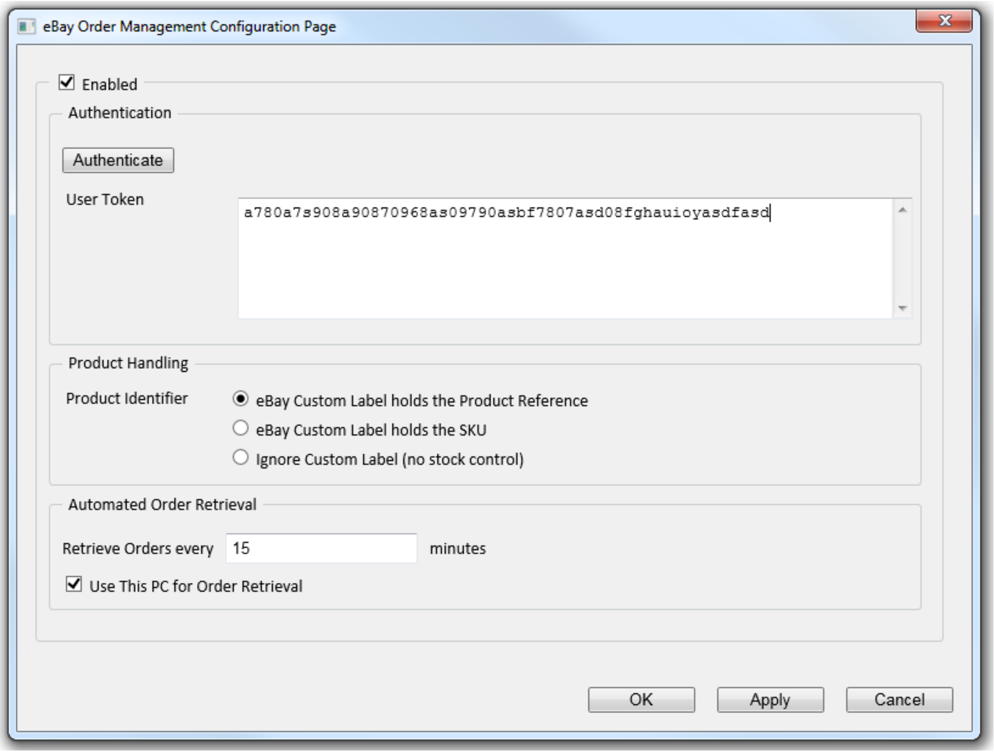
The user interface for configuration of Extensions must be placed in the Settings menu.

Some simple Extensions may not require a Configuration page. The Extension negotiates with SDD on start-up, specifying if a Configuration dialog is required.

In the Settings menu there is a menu option 'Extensions', with a submenu for each Extension:



Selecting an Extension opens the configuration dialog named for it. CSS is used to reproduce as far as possible the look and feel of the current SellerDeck Desktop dialogs.


eBay Order Management Configuration Page

☑ Enabled

Authentication

User Token: a780a7s908a90870968as09790asbf7807asd08fghauioyasdfasd

Product Handling

Product Identifier
● eBay Custom Label holds the Product Reference
○ eBay Custom Label holds the SKU
○ Ignore Custom Label (no stock control)

Automated Order Retrieval

Retrieve Orders every 15 minutes

☑ Use This PC for Order Retrieval

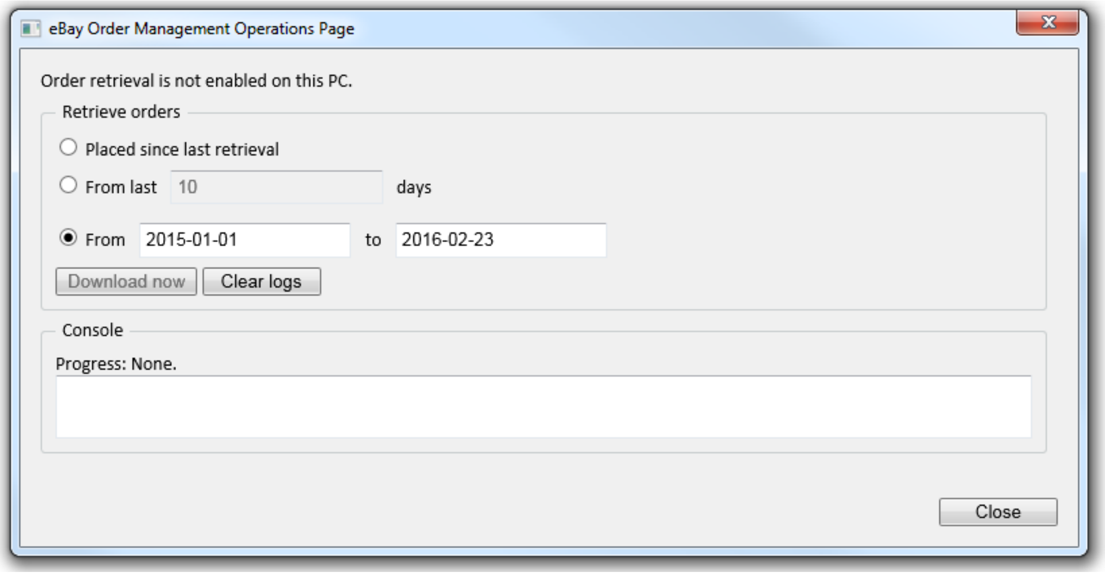
On multi user systems, only an Administrator will see the 'Extensions' option in the 'Settings' menu.

Some configuration e.g. of templates could be done via Layouts. For example this could contain the template for an Email. Any layout must be appropriately named using the Extension Prefix, or be contained in a folder in the Design Library named with the Extension Prefix.

3.8.3 Operation

The user interface for occasional manual operations of the Extension must be placed in the Operations menu. This operates in a similar way to the Configuration menu dialogs.

Think carefully about whether operations should be placed on the menu or in a main tab within the application. If the Extension normally operates by itself on a timer and occasional manual override, then a main tab should not be used.



Order retrieval is not enabled on this PC.

○ Placed since last retrieval

○ From last 10 days

● From 2015-01-01 to 2016-02-23

Progress: None.

3.8.4 Application Tab

If the Extension has operations that are used every day, or multiple different operations; or needs to display complex information for the user, then a main Application Tab should be used. These appear along with the other tabs in the main area of the product and allow more extensive interaction with the user.

Operations in this tab should be implemented using Ajax calls.

3.8.5 Support for Dialogs

SDD supplies a standard style sheet for pages shown in dialogs. This enforces a house style for UI elements which closely matches dialogs within the product. All Extension HTML files which are shown in dialogs should refer to this style sheet which will be referenced from `../../root/css/sde-dialog.css`.

The Extension can use additional privately-defined styles as required but won't be accepted if it strays too much from the house style.

The dialogs have a standard layout which, when combined with the CSS creates a scrollable main area for controls, with a fixed button bar across the bottom. For example dialogs see the Tutorial code, and look for `settings.php` and `help.html`.

The standard buttons are: `ButtonOK`, `ButtonApply` and `ButtonCancel`. SDD listens for clicks on OK and Cancel and closes the dialog. If OK is pressed it is up to the JavaScript on the page to make an Ajax call to the server to store the data. If validation of user settings is being applied and the values on the page may be wrong, then a dummy OK button can be shown, and the real ButtonOK click via a JavaScript call once all validation has passed; alternatively preventDefault can be called.

Main tabs have more flexibility and don't have to appear like dialogs.

3.8.6 Long running events

The library EventSource.js (https://github.com/Yaffle/EventSource) can be used for long-running events. This is included as standard in 16.0.2 and later. Note that standard EventSource API is not included in Internet Explorer so it has to be supplied by this polyfill.

3.8.7 Standard libraries

Version 16.0.0 and 16.0.1 of SDD include jQuery as standard.

Version 16.0.2 includes jQuery, jQuery-UI, Bootstrap and EventSource.js.

3.9 Site Files

Files from the current site can be referenced from web pages created by the Extension using the path `../../site/<filename>`. The main use for this is to show product images in the browser.

3.10 Timed Operations

SDD can create a timer function for Extensions. Timed operations e.g. download orders are under the control of desktop. On start-up SDD gets the configuration information from the Extension and starts the timers.

Timed operations are suspended whenever auto sync is suspended.

Timers are temporarily disabled on importing a snapshot to a different PC. This is done using the same mechanism and warning that prevents upload/download under the same circumstances.

3.11 Debugging

There are some facilities for debugging Extensions.

This is mostly driven from the command-line flag `-extdebug` for `catalog.exe` i.e.

```
catalog.exe -extdebug
```

When this is added, the normal right-click menu in the embedded browser is enabled. This allows the URL for the page e.g. an Extension dialog, or Extension tab, to be found (under Properties) and copied to the clipboard. The URL will work in a normal browser and developer tools can be used to see Ajax data and debug JavaScript etc.

In addition, if `W` (or `*` which turns on all debug-level tracing) is put in the Tracing Filter (in Help > Troubleshooting) (and if `-extdebug` is set) then all data transferred in the webserver is logged to the application trace file – default `C:\ProgramData\SellerDeck\SellerDeck 2016\Logs\CatalogTrace.txt`

This generates a lot of information including duplication of data received from the CGI application – once when received, and once when sending out to the browser – but can be useful.

The state of the `-extdebug` flag as well as the Tracing Filter string are sent through to the Extension as part of the handshaking process (see below). The SellerDeck PHP Extensions store this for configuring in-built tracing in the Extension, enabling debug and info level logging.

In addition the content of the Trace Filter is sent through. The Desktop only uses characters up to the first "." so text after that can be used by Extensions as they want. For example a trace filter text of ".sales:debug" could be used by a sales Extension to turn on debug data on the page.

4 Extension Development Process

All contact with SellerDeck should be via the email address sdd-extensions@sellerdeck.com. This is forwarded to a number of personnel within SellerDeck.

4.1 Steps

The normal steps for development of an Extension are:

1. Request Development EKB, including allocation of unique Extension Prefix
2. Develop and test the Extension
3. Request full EKB
4. Submit Extension to SellerDeck for acceptance. This must include a full Test Procedure
5. SellerDeck lists the Extension on the Extensions Catalogue

If an Extension is developed for a single customer then it can be provided directly to the customer. The customer must still purchase licence keys for the Extension through SellerDeck.

4.2 Extension Submission

4.2.1 Test Procedure

Any Extensions submitted for listing on the Extension Catalogue must include a full Test Procedure as a document. This must give all steps and data required to fully test the Extension with Pass/Fail status recorded.

It must be updated as new versions of the Extension are submitted. SellerDeck may update the Test Procedure and send it back to the Extension developer if particular issues are noted which should be rectified. Note that TortoiseSVN includes a facility to compare different versions of document files in Word. This is very useful for comparing different versions of the Test Procedure to see what has been changed.

An example Test Procedure is included in the Extension Development Kit.

SellerDeck will use this document to

- Check the operation of the Extension
- Validate the Extension against new releases of SDD
- Validate changes to the standard Extension Framework if the Extension is written in PHP

4.2.2 Source Code

SellerDeck also require the source code of the Extension. The Developer will retain Copyright.

The Source Code must include a `install.reg` file which sets up the registry keys for SDD, assuming that SDD is installed in the standard folder, and the source code for the Extension is in the folder `C:\SellerDeck\Extensions\<extensionname>`

Therefore interim releases of the Extension can be installed by SellerDeck staff by unzipping or copying the code to that folder then running the `install.reg` file.

The Developer is responsible for using suitable tools for maintaining the Source Code:

- It should be stored in a code repository such as Git or Subversion
- It must be neatly formatted and indented with tabs

- Commented out code must be removed – use the code repository to maintain different versions
- Each file must include a copyright notice in the header

Note that SellerDeck will commit the Source Code to its internal code repository. This will help us see what has changed between versions.

4.2.3 Installer

The Extension must be installed using a professional Windows installer package. SellerDeck uses Advanced Installer using .msi files. Liability disclaimers must be shown and installation refused if the disclaimer is not present.

SellerDeck will create the installer as part of the submission service for PHP Extensions.

If the Extension is implemented in PHP then Source Guardian (http://www.sourceguardian.com/) must be used to protect your code. Source Guardian should only be applied to the `app` folder as the other PHP e.g. the framework is freely available. That also limits the overhead of using source obfuscation.

4.2.4 Release Note

A Release Note file must be included in each Extension. The installer will show a link to it from the last page.

The Release Note must show:

- the current version of the Extension
- a short paragraph describing its functionality
- Next steps on setting up the Extension i.e. where to find further instructions in the SellerDeck Desktop UI
- A Changes section for each release, showing the changes made for that release. The latest release must be shown first.

The Release Note must also be accessible from the Help using an “About” button alongside the other buttons at the bottom of the screen.

For PHP Extensions the Release Note file must be stored alongside the help.html file in `\app\public\releasenote.html`. The file must use HTML for formatting but can’t use the standard CSS and other libraries installed within SellerDeck Desktop, as it will be opened from the file system directly from the Installer. Therefore it should use simple header, paragraph and list tags.

4.2.5 Submission Process

SellerDeck may charge for submission of an Extension. This will be for the time taken to check the Extension, including QA and Development time, as well as creation of the installer.

The process which will be followed by SellerDeck when an Extension is submitted is:

1. Test Procedure is committed to SVN along with the Extension code.
2. Test Procedure is followed by QA. The pass/fail status is updated in the document.
3. If QA discover any other failures not mentioned by the Test Procedure then they are added as additional tests
4. QA add the date to a numbered “Status” table including the number of tests, number of passes and number of failures
5. QA commit the Test Procedure to SVN

If there are any failures, then:

6. Test Procedure is sent to the Extension developer
7. Extension developer rectifies any failures
8. Test Procedure must be followed in its entirety and updated by the Extension developer with a new entry in the status table
9. Extension developer sends back new code along with new Test Procedure

The cycle continues with step 1. Multiple submissions may incur further charges by SellerDeck.

If a bug is found by the Extension developer then a suitable test for that should be added to the Test Procedure, and the code submitted again as above.

In communication with the Extension developer, all references to the Test Procedure should include the current number from the status table so at no time is anyone working from an old version of the document.

The installer will be created when the Extension has been accepted.

4.3 Extension Catalogue

SellerDeck provide an Extension Catalogue. SellerDeck Desktop 2016 has a tab showing the Extension Catalogue. This is in fact an Extension which simply redirects to the Extension Catalogue on the SellerDeck website.

Once your Extension has been accepted it will be listed in the Extensions Catalogue and will be available for download.

5 Forums

SellerDeck Forums have been created for Extensions. These can be found at http://community.sellerdeck.com/forumdisplay.php?f=161 .

5.1 General Extension Discussions

The forum for discussion about Extensions which doesn’t fit in any other category is http://community.sellerdeck.com/forumdisplay.php?f=165 .

5.2 Ideas and Requests

The forum for discussion about Ideas and Requests is at http://community.sellerdeck.com/forumdisplay.php?f=162 .

5.3 Extension Development

The forum for Extension development discussion is at http://community.sellerdeck.com/forumdisplay.php?f=163 .

5.4 Extension-Specific Forums

Each Extension that is available in the Extensions Catalogue has a forum created for it. This will be used for discussing new features and problems. Currently the Sales Dashboard Extension forum is at http://community.sellerdeck.com/forumdisplay.php?f=164 .

Appendix A Extension Configuration Requests

At startup a request is made to each Extension found in the registry. The ConfigURI is invoked in a POST request with the data as below.

The information sent to the Extension will be in an XML format and will include:

1. Version of SellerDeck
2. Version of request data

The xml response uses the following format

```
<?xml version="1.0" encoding="utf-8"?>
<request version="1">
        <versionid>
                <major>16</major>
                <minor>0</minor>
                <maintenance>0</maintenance>
                <build>DYNO</build>
        </versionid>
</request>
```

The response will include:

1. The version of the response data
2. The version of the Extension
3. The HEKB value allocated to the Extension

The xml response uses the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<response version="1" name="version-information">
        <status>success/failure</status>
        <statusmessage>Explanation of failure</statusmessage>
        <versionid>
                <major>1</major>
                <minor>0</minor>
                <maintenance>0</maintenance>
                <versionstring>Free Text</versionstring>
        </versionid>
        <requiredversion>
                <major>16</major>
                <minor>0</minor>
                <maintenance>0</maintenance>
        </requiredversion>
        <publisher>
                <name>SellerDeck Ltd</name>
                <url>http://www.sellerdeck.co.uk/sde/ebay</url>
        </publisher>
        <authorization>
                <key>HEKB</key>
        </authorization>
</response>
```

If the Extension is determined to be licenced, then another call will be made to the Extension. This is relevant for Extensions that use programs:

1. The API key to use.
2. The API port.
3. Database information
4. Location of data files

The xml request uses the following format. Note that `paths/logs/debug` is 1 if `-extdebug` is passed on the command line. Typically it should be used to enable more detailed Extension logging. `paths/logs/tracefilter` is the content of the Troubleshooting Trace Filter edit box and can be used to control Extension features as required. All characters after the first "." are ignored by the desktop.

```
<?xml version="1.0" encoding="utf-8"?>
<request version="1" name="configuration">
   <versionid>
      <major>16</major>
      <minor>0</minor>
      <maintenance>0</maintenance>
      <build>PECB</build>
      <orderversionid>25</orderversionid>
      <orderdetailversionid>12</orderdetailversionid>
   </versionid>
   <notificationsapi>

<apiurl>http://192.168.1.100:54385/:54385Invoke?Handler=ActinicNotificationService</ap
iurl>
      <apiwsdl>http://192.18.0.2:54385/ActinicNotificationService?wsdl</apiwsdl>
      <apiport>54385</apiport>
      <apikey>fD0uZTXjl09Fwv5BnoMqoDy2B3nB2NItJ</apikey>
   </notificationsapi>
   <baseurl>http://localhost:8080/5M3xlm2aXDSpw6RIvtZhcvw0e1bJHo/</baseurl>
   <extensionurl>extensions/tutorial/</extensionurl>
   <webserverkey>/5M3xlm2aXDSpw6RIvtZhcvw0e1bJHo</webserverkey>
   <site>
      <sitename>v16SQL</sitename>
      <loggedinusername>Administrator</loggedinusername>
      <loggedinuserid>1</loggedinuserid>
   </site>
   <database>
      <dbtype>MS Access or MS SQL Server</dbtype>
      <catalogdbfilepath>C:\Users\UserName\Documents\SellerDeck
2016\Sites\Site1\ActinicCatalog.mdb</catalogdbfilepath>
      <shippingdbfilepath>C:\Users\UserName\Documents\SellerDeck
2016\Sites\Site1\ShipControl\SimpleShipping.mdb</shippingdbfilepath>
      <catalogdbsqlname>v16SQL_Catalog</catalogdbsqlname>
      <shippingdbsqlname>v16SQL_Shipping</shippingdbsqlname>
      <servername>OPTIPLEX990\SQLSERVER2008R2</servername>
      <dbauthtype>DB or NT</dbauthtype>
      <username>v16SQL_User</username>
      <userpwd>S3cr3tPassw0rd</userpwd>
   </database>
   <paths>
      <files>
         <site>E:\Sites\2016\Site1</site>
         <temporary>C:\ProgramData\SellerDeck 2016\Sites\Site1\sde\Tutorial
1.0.0</temporary>
      </files>
      <logs>
         <site>C:\ProgramData\SellerDeck 2016\Logs\Sites\Site1\sde\Tutorial
1.0.0</site>
         <global>C:\ProgramData\SellerDeck 2016\Logs\sde\Tutorial 1.0.0</global>
         <debug>0</debug>
         <tracefilter>0</filter>
      </logs>
   </paths>
</request>
```

The response will include:

1. The status of the request

```
<?xml version="1.0" encoding="utf-8"?>
<response version="1" name="version-information">
        <status>success/failure</status>
        <statusmessage>Explanation of failure</statusmessage>
```

```
</response>
```

```
<?xml version="1.0" encoding="utf-8"?>
<request version="1" name="extension-configuration" />
```

On success SellerDeck Desktop sends a request to get the Extension configuration

The response will include:

1. The status of the request
2. Whether a tab is required, and if so, the root file to serve the tab.
3. Whether a Help page is required, and if so, the root file to serve the page.
4. Whether a Configuration page is required, and if so, the root file to serve the page, the width and height of the configuration dialog
5. The Publisher of the Extension e.g. SellerDeck Ltd.
6. The URL for the Extension e.g. http://www.sellerdeck.co.uk/sde/ebay .
7. The disabled order function configuration: the channel name associated with the Extension, and comma separated IDs of the disabled order functions . At the time of writing the following IDs are accepted

    1 - complete customers tab

    2 - complete taxes tab

    3 - complete totals tab

    4 - complete mail tab

    5 - line items tab: Cancel, Add New Item, Add Adjustment, Increase Qty and Decrease Qty; Order Line dialog: Cancelled and Unit Price should be disabled. Shipped Qty once applied

    6 - shipping and handling tab: Update Order Shipping and Update Order Handling buttons

    7 - payments tab: all the controls except View Payments and Copy Order Number buttons

    8 – Backordering: disable backordering

```
<?xml version="1.0" encoding="utf-8"?>
<response version="1" name="configuration">
        <status>success/failure</status>
        <statusmessage>Explanation of failure</statusmessage>
        <configuration>
                <timer>
                        <interval>23</interval>
                        <timerurl>timercgi.php</timerurl>
                </timer>
                <extensionstab>
                        <requirestab>true</requirestab>
                        <displayname>eBay Orders</displayname>
                <extensionsindexfile>index.php</extensionsindexfile>
                </extensionstab>
                <helppage>
                        <requireshelp>true</requireshelp>
                        <helpfile>help.html</helpfile>
                        <width>200</width>
                        <height>400</height>
                </helppage>
                <configpage>
                        <requiresconfig>true</requiresconfig>
                        <configindexfile>setup.php</configindexfile>
                        <width>200</width>
                        <height>400</height>
                </configpage>
                <operationspage>
                        <requiresoperations>true</requiresoperations>
                        <operationsindexfile>operations.php</operationsindexfile>
                        <width>200</width>
                        <height>400</height>
                </operationspage>
                <disabledorderfunctions>
                        <channelname>eBay</channelname>
                        <csv>1,2</csv>
                </disabledorderfunctions>
        </configuration>
</response>
```

Appendix B SellerDeck API

A single SOAP method 'InvokeApi' is available in SDD accepting XML as request and response. Any error message also be in XML.

There are a number of different API calls available, each with different XML.

These APIs are made on the local machine. The configuration exchange outlined in Appendix A provides information about the URL for the API as well as the WSDL location.

If there are multiple instances of SellerDeck installed then they work with the Sync Server to coordinate actions across all instances. Otherwise they just affect the local instance.

Each API call given here includes the XML Schema definition used to validate the basic request format.

B.1 Lock Request Format

B.1.1 Request XML Design

Lock request to the SDD used to claim an item within a group for specific action. The below format is used for the lock request. The lock can be single or all. Group id is mandatory for lock request. Item reference can be empty when all items are requested to be locked.

```
<?xml version="1.0" encoding="UTF-8"?>
<request version="1">
   <api>sync</api>
   <apikey>fD0uZTXjl09Fwv5BnoMqoDy2B3nB2NItJ</apikey>
   <command>
      <type>lock</type>
      <mode>single/all</mode>
      <parameters>
         <item>
            <group>order</group>
            <reference>DEMO0000000002</reference>
         </item>
      </parameters>
   </command>
</request>
```

Mode 'single' does the job of locking single item. However, 'all' does lock all items in the specified group. The node 'reference' can be empty for locking all items.

Group can be any of the following:

```
invalid
setting
order
content
usersoperation
license
usertype
customeraccount
generalsetting
rtsgroup
resource
```

Below table shows the sample Group Id and reference that can be used for locking an item in a group.

No	Group	Reference
1	order	Order Number (used to lock an order)
2	license	Key (check for logon key is unique)
3	usersoperation	User Name (used while creating and deleting an user)
4	usertype	Administrator (to check if we can lock as administrator)
5	customeraccount	Customer Name (used while editing customer account)
6	generalsetting	"Duplicates" (unusable, used to do a quick claim release to indicate an edit to other machines)
7	setting	Settings Id (use the below reference to claim/release to notify other users that we have changed preferences PREFERENCES_LOCK_REF)
8	content	Product Reference
9	resource	Private named resource (used to lock/unlock of any named resource)

Note:

Content Group for locking Section/Brochures need special handling (yet to implement).

B.1.2 Response XML Design

The response XML has complete information about the API, command with the returns.

Below XML format used for response to the lock command.

```
<?xml version="1.0"?>
<response version="1">
   <api>sync</api>
   <command>lock</command>
   <status>success/failure</status>
   <mode>single</mode>
   <returns>
      <item>
         <group></group>
         <reference></reference>
         <message>locked</message>
         <returnid>0</returnid>
      </item>
   </returns>
</response>
```

B.2 Unlock Request Format

B.2.1 Request XML Design

Unlock request is used to release the claimed item so that the item can be used for some other operation. Below format is used to unlock an item.

When an item is unlocked it automatically notifies other instances of SDD about the change forcing an update of caches.

```
<?xml version="1.0" encoding="UTF-8"?>
<request version="1">
   <api>sync</api>
   <apikey>D2n3Fj32ez2Pv5QslNpdUR9PDxN1dHBV0</apikey>
   <command>
      <type>unlock</type>
      <mode>single</mode>
      <parameters>
         <item>
            <group>order</group>
            <editaction>edited</editaction>
            <reference>DEMO0000000003</reference>
         </item>
      </parameters>
   </command>
</request>
```

Edit action might be any of the below. Default is '`edited`'

```
  undefined
  noedit
  claimed
  edited
  deleted
  new
  subitemsresequenced
  purged
```

B.2.2 Response XML Design

Below format can be followed for unlock command response:

```
<?xml version="1.0"?>
<response version="1">
   <api>sync</api>
   <command>unlock</command>
   <status>success/failure</status>
   <mode>single</mode>
   <returns>
      <item>
         <group></group>
         <editaction></editaction>
         <reference></reference>
         <message>unlocked</message>
         <returnid>0</returnid>
      </item>
   </returns>
</response>
```

B.3 Lock/Unlock Schema

Schema design for supporting lock/unlock command is given below:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <!-- request definitions  -->
   <xs:element name="request">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="api" type="apiType" />
            <xs:element name="apikey" type="typeApiKey" />
            <xs:element name="command">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="type" type="typeType" />
                     <xs:element name="mode" type="modeType" />
                     <xs:element name="parameters">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="item">
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="group" type="groupType" />
                                       <xs:element name="editaction"
type="editactionType" minOccurs="0"/>
                                       <xs:element name="reference" type="xs:string"
/>
                                    </xs:sequence>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <!-- version check -->
         <xs:attribute name="version" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:decimal">
                  <xs:pattern value="\d+"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <!-- group Type -->
   <xs:simpleType name="groupType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="invalid"/>
         <xs:enumeration value="setting"/>
         <xs:enumeration value="order"/>
         <xs:enumeration value="content"/>
         <xs:enumeration value="usersoperation"/>
         <xs:enumeration value="license"/>
         <xs:enumeration value="usertype"/>
         <xs:enumeration value="customeraccount"/>
         <xs:enumeration value="generalsetting"/>
         <xs:enumeration value="rtsgroup"/>
         <xs:enumeration value="resource"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- editaction Type -->
   <xs:simpleType name="editactionType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="undefined"/>
         <xs:enumeration value="noedit"/>
         <xs:enumeration value="claimed"/>
         <xs:enumeration value="edited"/>
         <xs:enumeration value="deleted"/>
         <xs:enumeration value="new"/>
         <xs:enumeration value="subitemsresequenced"/>
         <xs:enumeration value="purged"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- mode Type -->
   <xs:simpleType name="modeType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="single"/>
         <xs:enumeration value="all"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- type Type -->
   <xs:simpleType name="typeType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="lock"/>
         <xs:enumeration value="unlock"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- api Type -->
   <xs:simpleType name="apiType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="sync"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- type ApiKey -->
   <xs:simpleType name="typeApiKey">
      <xs:restriction base="xs:string">
```

```
         <xs:minLength value="33"/>
         <xs:maxLength value="33"/>
         <xs:pattern value="[a-zA-Z0-9]*"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

New types such as groupidType, modeType etc are defined to apply specific restrictions on the input tags that are accepted in the lock/unlock request XML. The schema is validated when loading the XML for further processing, and validation error is thrown as an error XML output.

B.4 Stock Update

B.4.1 Request XML Design

The **Stock Update API** request is used to notify of any stock changes. This updates the database and creates the correct information for the website. The format is below:

```
<?xml version="1.0" encoding="UTF-8"?>
<request version="1">
   <api>stock</api>
   <apikey>fD0uZTXjl09Fwv5BnoMqoDy2B3nB2NItJ</apikey>
   <command>
      <type>update</type>
      <parameters>
         <product>
            <mode>absolute</mode>
            <prodref>1</prodref>
            <stocktoupdate>80</stocktoupdate>
         </product>
         <product>
            <mode>relative</mode>
            <prodref>2</prodref>
            <stocktoupdate>10</stocktoupdate>
         </product>
      </parameters>
   </command>
</request>
```

Response XML is loaded by XML response classes as template and edited with the appropriate response.

B.4.2 Schema

Schema design for supporting stock method is given below:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="request">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="api" type="apiType" />
            <xs:element name="apikey" type="typeApiKey" />
            <xs:element name="command">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="type" type="typeType" />
                     <xs:element name="parameters">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element maxOccurs="unbounded" name="product">
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="mode" type="modeType" />
                                       <xs:element name="prodref" type="xs:string"
/>
```

```
                                       <xs:element name="stocktoupdate"
type="stocktoupdateType" />
                                    </xs:sequence>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <!-- version check -->
         <xs:attribute name="version" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:decimal">
                  <xs:pattern value="\d+"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <!-- api Type -->
   <xs:simpleType name="apiType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="stock" />
      </xs:restriction>
   </xs:simpleType>
   <!-- type ApiKey -->
   <xs:simpleType name="typeApiKey">
      <xs:restriction base="xs:string">
         <xs:minLength value="33"/>
         <xs:maxLength value="33"/>
         <xs:pattern value="[a-zA-Z0-9]*"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- type Type -->
   <xs:simpleType name="typeType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="update" />
      </xs:restriction>
   </xs:simpleType>
   <!-- mode Type -->
   <xs:simpleType name="modeType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="absolute"/>
         <xs:enumeration value="relative"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- stocktoupdate Type -->
   <xs:simpleType name="stocktoupdateType">
      <xs:restriction base="xs:integer">
         <xs:minInclusive value="1"/>
         <xs:maxInclusive value="32767"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

B.5 SDE Configuration Reload

B.5.1 Request Format

SDE Configuration Reload request to the SDD used to reload the SDE configuration which might be modified by SDE. The format is given below:

```
<!-- sde configuration reload -->
<?xml version="1.0" encoding="UTF-8"?>
<request version="1">
   <api>sdd</api>
   <apikey>fD0uZTXjl09Fwv5BnoMqoDy2B3nB2NItJ</apikey>
   <command>
      <type>sdereloadconfig</type>
   </command>
</request>
```

B.5.2 Response XML Design

The response XML is below:

```
<?xml version="1.0"?>
<response version="1">
   <api>sdd</api>
   <command>sdereloadconfig</command>
   <status>success</status>
</response>
```

B.5.3 Schema

Schema design for supporting the configuration reload command is below:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <!-- request definitions  -->
   <xs:element name="request">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="api" type="apiType" />
            <xs:element name="apikey" type="typeApiKey" />
            <xs:element name="command">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="type" type="typeType" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <!-- version check -->
         <xs:attribute name="version" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:decimal">
                  <xs:pattern value="\d+"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <!-- type Type -->
   <xs:simpleType name="typeType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="sdereloadconfig"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- api Type -->
   <xs:simpleType name="apiType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="sdd"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- type ApiKey -->
   <xs:simpleType name="typeApiKey">
      <xs:restriction base="xs:string">
         <xs:minLength value="33"/>
         <xs:maxLength value="33"/>
         <xs:pattern value="[a-zA-Z0-9]*"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

New types such as typeType, apiType etc are defined to apply specific restrictions on the input tags that are accepted in display request XML. The schema will be validated when loading the XML for further processing, and validation error is thrown as an error XML output.

B.6 Update Order List

Display order request to the SDD used to add orders to the Order form of SDD.

B.6.1 Request format

The below format is used for the request.

```xml
<!-- display order -->
<?xml version="1.0" encoding="UTF-8"?>
<request version="1">
   <api>order</api>
   <apikey>fD0uZTXjl09Fwv5BnoMqoDy2B3nB2NItJ</apikey>
   <command>
      <type>display</type>
      <parameters>
         <order>
            <reference>DEMO0000000001</reference>
         </order>
         <order>
            <reference>DEMO0000000002</reference>
         </order>
         <order>
            <reference>DEMO0000000003</reference>
         </order>
      </parameters>
   </command>
</request>
```

B.6.2 Response XML Design

Below format is followed for the command response:

```xml
<?xml version="1.0"?>
<response version="1">
   <api>order</api>
   <status></status>
   <returns>
      <order>
         <reference>DEMO0000000001</reference>
         <message>Already in the list</message>
      </order>
      <order>
         <reference>DEMO0000000002</reference>
         <message>Already in the list</message>
      </order>
      <order>
         <reference>DEMO0000000003</reference>
         <message>Already in the list</message>
      </order>
   </returns>
</response>
```

B.6.3 Schema

The Schema design for supporting command validation is below:

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <!-- request definitions  -->
   <xs:element name="request">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="api" type="apiType" />
            <xs:element name="apikey" type="typeApiKey" />
            <xs:element name="command">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="type" type="typeType" />
                     <xs:element name="parameters">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element maxOccurs="unbounded" name="order">
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="reference" type="xs:string"
/>
                                    </xs:sequence>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <!-- version check -->
         <xs:attribute name="version" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:decimal">
                  <xs:pattern value="\d+"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
      </xs:complexType>
   </xs:element>
   <!-- type Type -->
   <xs:simpleType name="typeType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="display"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- api Type -->
   <xs:simpleType name="apiType">
      <xs:restriction base="xs:string">
         <xs:enumeration value="order"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- type ApiKey -->
   <xs:simpleType name="typeApiKey">
      <xs:restriction base="xs:string">
         <xs:minLength value="33"/>
         <xs:maxLength value="33"/>
         <xs:pattern value="[a-zA-Z0-9]*"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```