Customization Manager Documentation

Release 4.0.4

Poplar Development

Aug 16, 2021

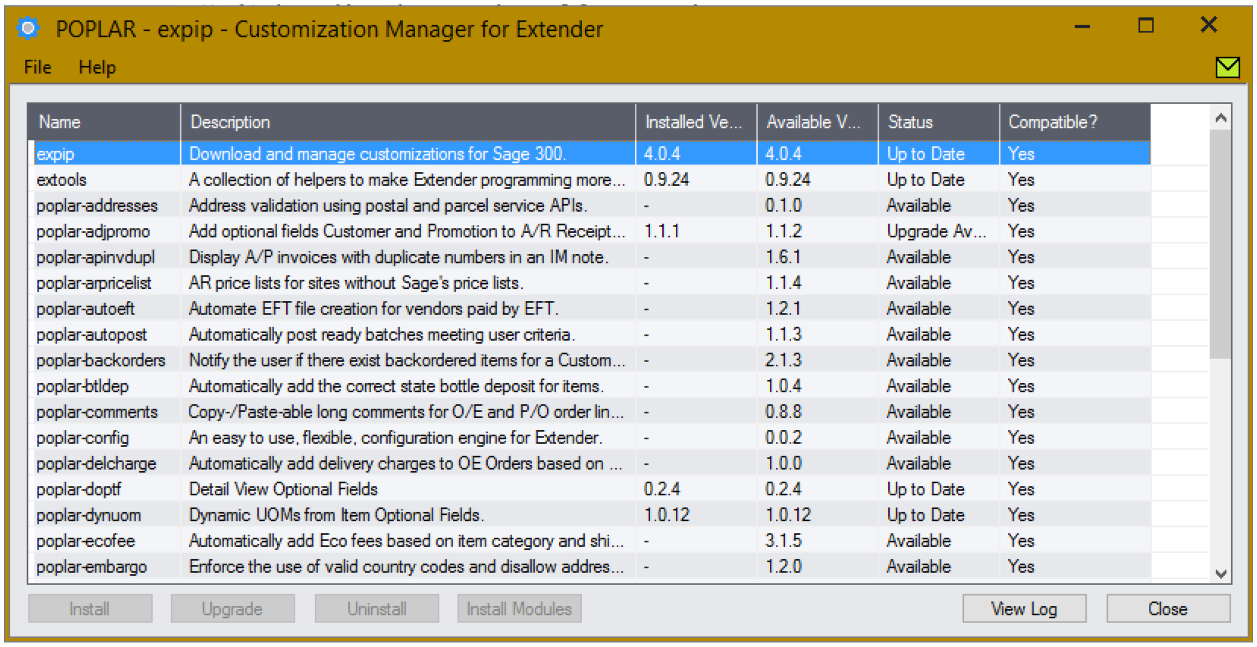
Contents:

Customization Manager for Orchid Extender unlocks the full potential of Python for customizing Sage 300. It enables quick and easy management of the embedded Python installation used by Extender and supports installation of most of the 200,000+ packages from the Python Packaging Index or any custom package from a version control system (VCS) repository.

`Download Customization Manager`

Note: Customization Manager version 9.0.0 has been released with full support for Extender 9 and Python 3.8.8. After upgrading to Extender 9 and activating the Python 3.8.8 environment you will need to reinstall all customizations managed by Customization Manager.

Many of the customizations currently distributed with Customization Manager are listed in the Customization Catalogue.



Name	Description	Installed Ve...	Available V...	Status	Compatible?
expip	Download and manage customizations for Sage 300.	4.0.4	4.0.4	Up to Date	Yes
extools	A collection of helpers to make Extender programming more...	0.9.24	0.9.24	Up to Date	Yes
poplar-addresses	Address validation using postal and parcel service APIs.	-	0.1.0	Available	Yes
poplar-adjpromo	Add optional fields Customer and Promotion to A/R Receipt...	1.1.1	1.1.2	Upgrade Av...	Yes
poplar-apinvdupl	Display A/P invoices with duplicate numbers in an IM note.	-	1.6.1	Available	Yes
poplar-arpricelist	AR price lists for sites without Sage's price lists.	-	1.1.4	Available	Yes
poplar-autoeft	Automate EFT file creation for vendors paid by EFT.	-	1.2.1	Available	Yes
poplar-autopost	Automatically post ready batches meeting user criteria.	-	1.1.3	Available	Yes
poplar-backorders	Notify the user if there exist backordered items for a Custom...	-	2.1.3	Available	Yes
poplar-btldep	Automatically add the correct state bottle deposit for items.	-	1.0.4	Available	Yes
poplar-comments	Copy-/Paste-able long comments for O/E and P/O order lin...	-	0.8.8	Available	Yes
poplar-config	An easy to use, flexible, configuration engine for Extender.	-	0.0.2	Available	Yes
poplar-delcharge	Automatically add delivery charges to OE Orders based on ...	-	1.0.0	Available	Yes
poplar-doptf	Detail View Optional Fields	0.2.4	0.2.4	Up to Date	Yes
poplar-dynuom	Dynamic UOMs from Item Optional Fields.	1.0.12	1.0.12	Up to Date	Yes
poplar-ecofee	Automatically add Eco fees based on item category and shi...	-	3.1.5	Available	Yes
poplar-embargo	Enforce the use of valid country codes and disallow addres...	-	1.2.0	Available	Yes

CHAPTER 1

How-Tos

This is a collection of guides detailing the steps to carry out common tasks with Customization Manager.

1.1 Install and Configure Customization Manager

Customization Manager is distributed as an Orchid Extender module for Sage 300. Once installed using the *Extender –> Setup –> Modules* screen and configured through *Extender –> Setup –> Custom Table Editor*, it can be used to install and manage customizations to Sage 300.

If you haven't already, download the most recent version of the Customization Manager module before getting started.

`Download Customization Manager`

1.1.1 Installation Video

This video walks through all the steps documented below in case you prefer a video.

1.1.2 Install the Module

Start by installing the Customization Manager module with Extender.

1. Open the *Extender –> Setup –> Modules* screen.
2. Click *Import* and select the `EXPIP.poplar.vi` file.
3. Once the module has imported, close the *Modules* screen.

1.1.3 Configure the Module

Customization Manager uses API keys to keep customizations private. Configure your API key:

1. Open the *Extender –> Setup –> Custom Table Editor* screen.

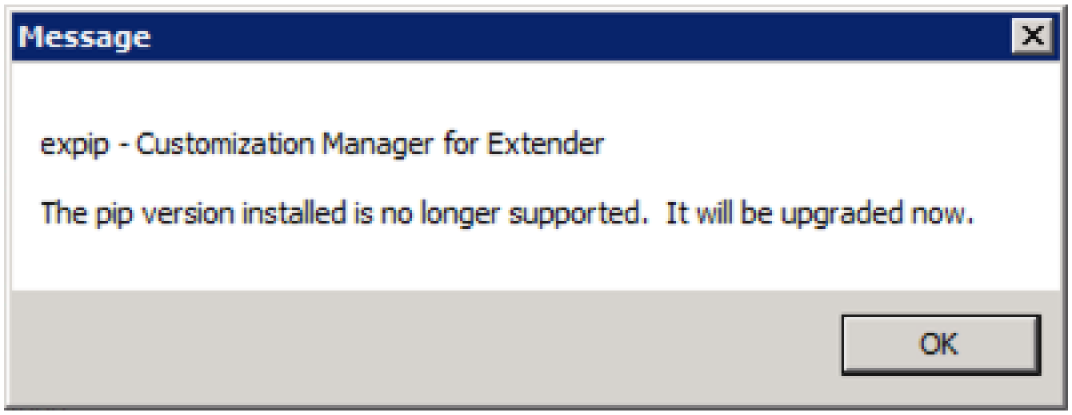
2. From the select box, select *Customization Manager* and click *Load*.
3. Input the API key and click *Add*.
4. Close the *Custom Table Editor*.

Note: Before starting Customization Manager for the first time, restart the Sage desktop to register new components.

1.1.4 Start Customization Manager for the First Time

When Customization Manager is started for the first time, it upgrades the pip package. The default version of `pip` that ships with Extender only supports an out-dated encryption protocol that is no longer used, making it impossible to install new packages securely.

Upgrading `pip` can take some time. First, the customization tries to perform an in-place upgrade without downloading new packages.

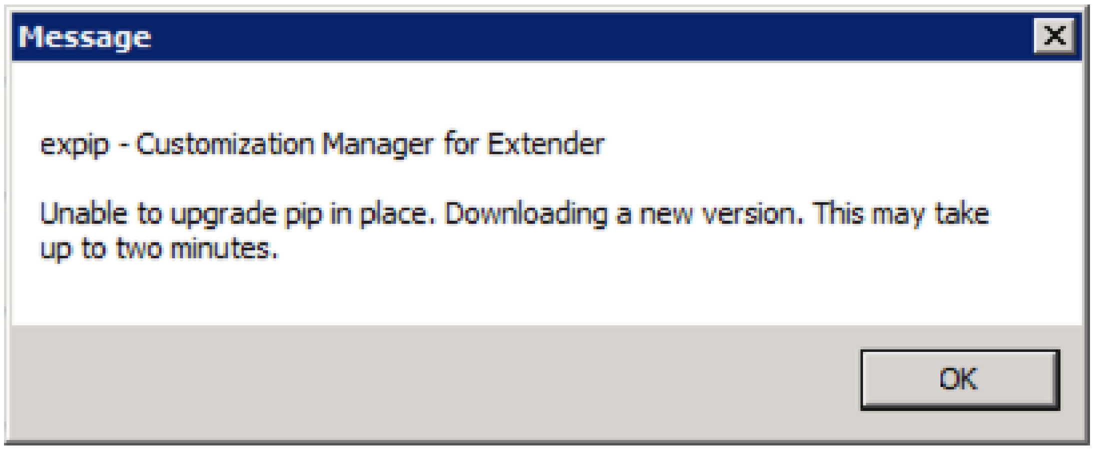


expip - Customization Manager for Extender

The pip version installed is no longer supported. It will be upgraded now.

Fig. 1: Before trying an in-place upgrade, this message is displayed.

If this isn't possible, a new version is downloaded and installed.



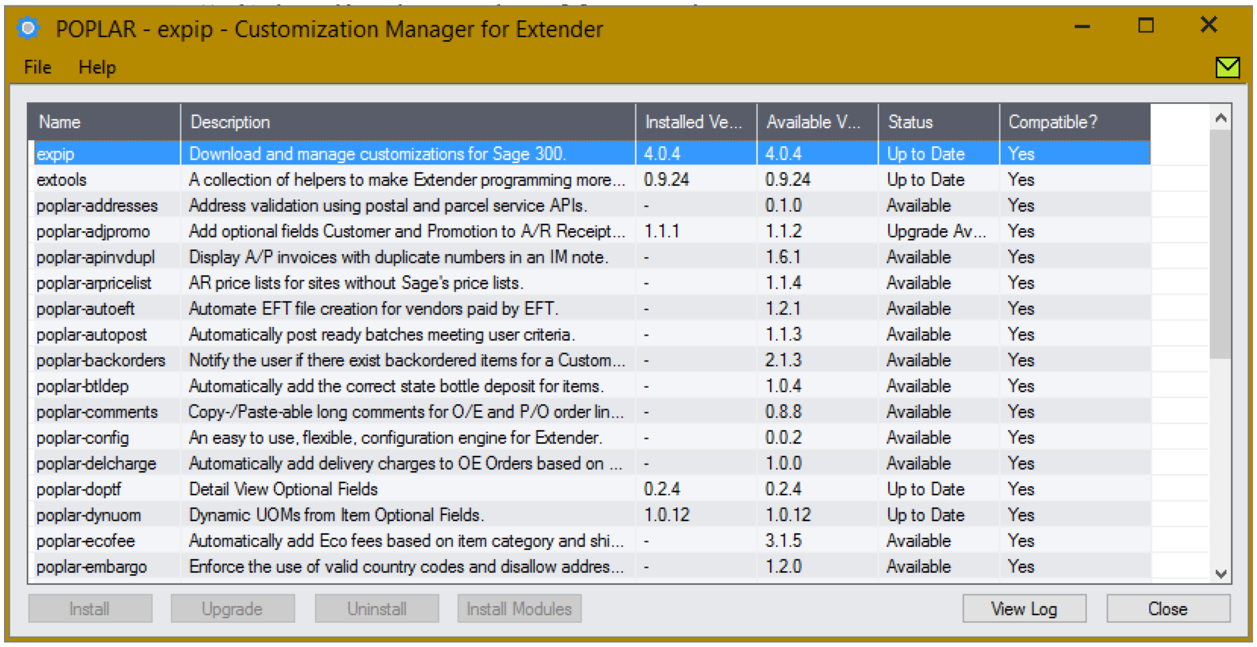
expip - Customization Manager for Extender

Unable to upgrade pip in place. Downloading a new version. This may take up to two minutes.

Fig. 2: If an in-place upgrade isn't possible, a new version is downloaded.

Together these steps can take up to three minutes. But the upgrade is only required on first start - you won't see it again!

With the upgrade complete, the Customization Manager screen opens.



Name	Description	Installed Ve...	Available V...	Status	Compatible?
expip	Download and manage customizations for Sage 300.	4.0.4	4.0.4	Up to Date	Yes
extools	A collection of helpers to make Extender programming more...	0.9.24	0.9.24	Up to Date	Yes
poplar-addresses	Address validation using postal and parcel service APIs.	-	0.1.0	Available	Yes
poplar-adjpromo	Add optional fields Customer and Promotion to A/R Receipt...	1.1.1	1.1.2	Upgrade Av...	Yes
poplar-apinvdupl	Display A/P invoices with duplicate numbers in an IM note.	-	1.6.1	Available	Yes
poplar-arpricelist	AR price lists for sites without Sage's price lists.	-	1.1.4	Available	Yes
poplar-autoeft	Automate EFT file creation for vendors paid by EFT.	-	1.2.1	Available	Yes
poplar-autopost	Automatically post ready batches meeting user criteria.	-	1.1.3	Available	Yes
poplar-backorders	Notify the user if there exist backordered items for a Custom...	-	2.1.3	Available	Yes
poplar-btldep	Automatically add the correct state bottle deposit for items.	-	1.0.4	Available	Yes
poplar-comments	Copy-/Paste-able long comments for O/E and P/O order lin...	-	0.8.8	Available	Yes
poplar-config	An easy to use, flexible, configuration engine for Extender.	-	0.0.2	Available	Yes
poplar-delcharge	Automatically add delivery charges to OE Orders based on ...	-	1.0.0	Available	Yes
poplar-doptf	Detail View Optional Fields	0.2.4	0.2.4	Up to Date	Yes
poplar-dynuom	Dynamic UOMs from Item Optional Fields.	1.0.12	1.0.12	Up to Date	Yes
poplar-ecofee	Automatically add Eco fees based on item category and shi...	-	3.1.5	Available	Yes
poplar-embargo	Enforce the use of valid country codes and disallow addres...	-	1.2.0	Available	Yes

1.1.5 Install the Customization Manager Package

Customization Manager can manage itself. By installing the packaged version of the module, you'll be able to upgrade it at the click of a button - no more file downloads. Installing the Customization Manager package is highly recommended.

1. Highlight the *expip* package.
2. Click the *Install* button.

All done. When new verions are released, you will see an upgrade available for the `expip` package.

1.2 Work with Customization Manager

Customization Manager is intended to be a simple and easy to use but it never hurts to have a few pointers when getting started.

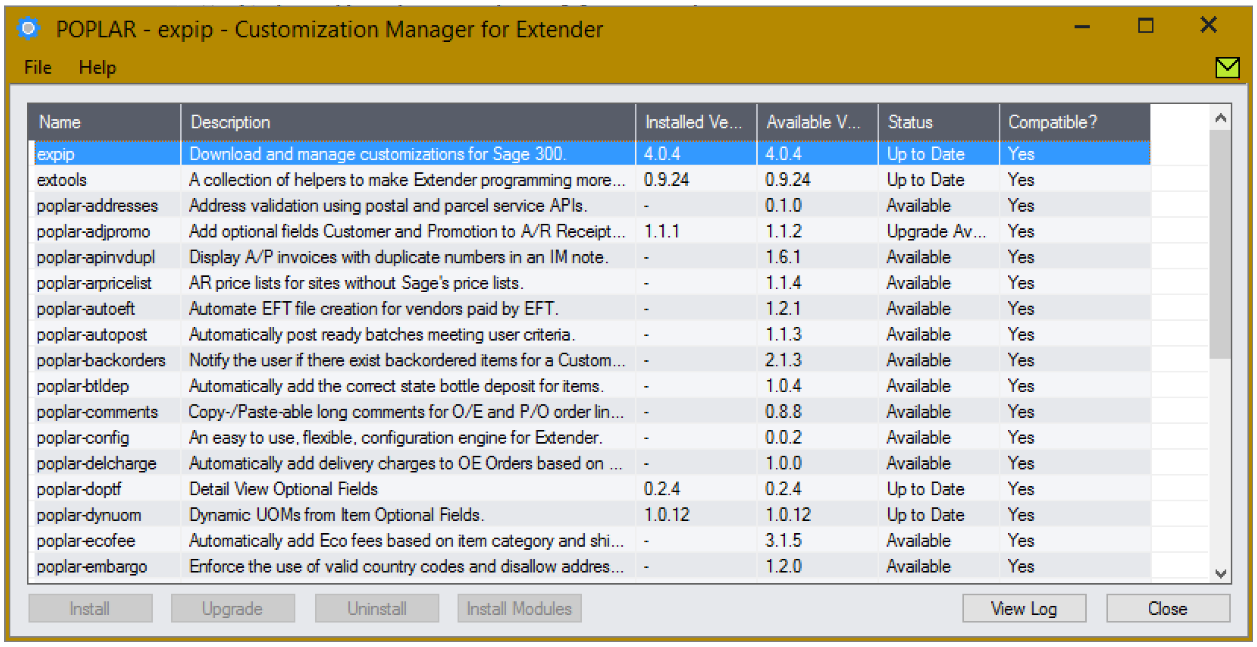
The interface is composed of the **Package Grid** and a set of **Action Buttons**.

1.2.1 The Pakage Grid

The package grid always displays all the customizations that are associated with the configured API key, whether they are installed or not.

All customizations have a short *Name* that is used to uniquely identify them. In the figure above, the first customization is named `expip` and is Customization Manager.

The name is followed by a meaningful *Description* of what the customization does.



Name	Description	Installed Ve...	Available V...	Status	Compatible?
expip	Download and manage customizations for Sage 300.	4.0.4	4.0.4	Up to Date	Yes
extools	A collection of helpers to make Extender programming more...	0.9.24	0.9.24	Up to Date	Yes
poplar-addresses	Address validation using postal and parcel service APIs.	-	0.1.0	Available	Yes
poplar-adjpromo	Add optional fields Customer and Promotion to A/R Receipt...	1.1.1	1.1.2	Upgrade Av...	Yes
poplar-apinvdupl	Display A/P invoices with duplicate numbers in an IM note.	-	1.6.1	Available	Yes
poplar-arpricelist	AR price lists for sites without Sage's price lists.	-	1.1.4	Available	Yes
poplar-autoeft	Automate EFT file creation for vendors paid by EFT.	-	1.2.1	Available	Yes
poplar-autopost	Automatically post ready batches meeting user criteria.	-	1.1.3	Available	Yes
poplar-backorders	Notify the user if there exist backordered items for a Custom...	-	2.1.3	Available	Yes
poplar-btldep	Automatically add the correct state bottle deposit for items.	-	1.0.4	Available	Yes
poplar-comments	Copy-/Paste-able long comments for O/E and P/O order lin...	-	0.8.8	Available	Yes
poplar-config	An easy to use, flexible, configuration engine for Extender.	-	0.0.2	Available	Yes
poplar-delcharge	Automatically add delivery charges to OE Orders based on ...	-	1.0.0	Available	Yes
poplar-doptf	Detail View Optional Fields	0.2.4	0.2.4	Up to Date	Yes
poplar-dynuom	Dynamic UOMs from Item Optional Fields.	1.0.12	1.0.12	Up to Date	Yes
poplar-ecofee	Automatically add Eco fees based on item category and shi...	-	3.1.5	Available	Yes
poplar-embargo	Enforce the use of valid country codes and disallow addres...	-	1.2.0	Available	Yes

Fig. 3: Customization Manager screen with the Package Grid above the Action Buttons.

There are two versions in the grid: the *Installed Version* and the *Available Version*. The *Installed Version* is the version running in the current company; it is set to – for customizations that are not installed. The *Available Version* is the latest version available for install.

All customizations have a *Status*. The status can be any of:

- `Available`: the customization is compatible and available for installation.
- `Up to Date`: the customization is installed and at the latest version.
- `Upgrade Available`: a newer version of the customization is available for download.

Finally, Customization Manager checks each customization for *Compatibility* with the current versions of Sage and Extender. If the Sage and Extender versions meet the minimum requirements, `Yes` is displayed. If they do not meet the minimum requirements, the compatiblity issue, either a Sage or Extender version, is displayed here.

1.2.2 The Action Buttons

The actions buttons are enabled and disabled based on the highlighted customization. There are two that never change:

- *Close*: close Customization Manager without taking any further action.
- *View Log*: open the log files from the two most recent actions and display them in the default text editor.

All the others change state based on whether the highlighted customization is installed, has an upgrade available, and is compatible.

- *Install*: *Install a Customization*
- *Upgrade*: *Upgrade a Customization*
- *Uninstall*: *Uninstall a Customization*
- *Install Modules*: for sites with multiple companies, *Install Modules*

1.3 Install a Customization

Installing a customization with Customization Manager is easy:

1. Open the *Extender –> Setup –> Customization Manager* screen.
2. A grid displays all the customizations that have been built for you. Highlight the customization and click *Install*.
3. A message is displayed with the result.
4. Close the Customization Manager window.

All done. Restart the Sage Desktop to allow Extender to register the changes.

1.3.1 The Install Button is disabled

The *Install* button will be disabled if:

- the customization is already installed, in which case its status will be either `Up to date` or `Upgrade Available`;
- the current Sage or Extender versions are not compatible with the customization, the compatibility field will indicate which versions are required.

If you're having trouble installing a customization, contact us.

1.4 Upgrade a Customization

Upgrading a customization with Customization Manager is easy:

1. Open the *Extender –> Setup –> Customization Manager* screen.
2. Highlight the customization row. The *Status* column will show `Upgrade Available` if a newer version can be installed.
3. Click the *Upgrade* button.
4. A message is displayed displaying the result.
5. Close the Customization Manager window.

The customzation is now upgraded. Restart the Sage desktop to allow Extender to register new components.

1.4.1 The Upgrade Button is disabled

The *Upgrade* button will be disabled if:

- the customization is not yet installed, in which case its status will be `Available`;
- the current Sage or Extender versions are not compatible with the customization, the compatibility field will indicate which versions are required.

If you're having trouble installing modules for a customization, contact us.

1.5 Uninstall a Customization

Uninstalling a customization is as easy as installing one:

1. Open the *Extender –> Setup –> Customization Manager* screen.
2. Highlight the customization and click *Uninstall*.
3. A message is displayed with the result.
4. Close the Customization Manager window.

Done.

1.5.1 The Uninstall Button is disabled

The *Uninstall* button will be disabled if the customization is not installed!

If you're having trouble uninstalling a customization, contact us.

1.6 Install Modules

In a multi-Company environment, the module files that enable a customization are only enabled in the Company from which the customization is first installed. The customization package is installed system wide and is easily enabled in other companies.

To enable a customization in other companies, you will need to install the customization modules:

1. Connect to the Company the customization needs installing in.
2. Open the *Extender –> Setup –> Customization Manager* screen.
3. A grid displays all the customizations that have been built for you. Highlight the customization and click *Install Modules*.
4. A message is displayed with the result.
5. Close the Customization Manager window.

Done. Restart the Sage Desktop to allow Extender to register the changes in this company.

1.6.1 The Install Modules Button is disabled

The *Install Modules* button will be disabled if:

- the customization is not yet installed, in which case its status will be `Available`;
- the current Sage or Extender versions are not compatible with the customization, the compatibility field will indicate which versions are required.

If you're having trouble installing modules for a customization, contact us.

1.7 Upgrade to Customization Manager 4

Customization Manager 4.0.0 adds a number of new features, such as postinstall scripts, test data fixture support, and improved version detection. It also has an improved interface that makes managing customizations easier.

If you're running an older version of Customization Manager, you probably installed it from a module file. Once installed from a module file, Customization Manager can *upgrade itself*. This is the easiest path to upgrade.

If you'd rather install the module file again manually, see the instructions in the *Upgrade with Module File* section instead.

1.7.1 Upgrade with Customization Manager

To get Customization Manager to manager and upgrade itself, install the `expip` package:

1. Open Customization Manager.
2. Select the `expip` package.
3. Click *Install* or *Upgrade*.

That's it. Customization Manager is now fully upgraded and the Desktop environment fixed up.

1.7.2 Upgrade with Module File

When upgrading from a previous version of Customization Manager, a new icon will be created on the desktop. For users that are upgrading, this will result in two identical icons, only one of which will work.

To avoid this situation, remove the existing icon before performing the upgrade. To remove the existing icon:

1. Navigate to *Extender –> Setup –> Scripts*.
2. Highlight the `EXPIP_poplar.py` file. Right click and select *Add to Desktop*.
3. A window with the desktop tree is displayed. Expand *Extender –> Setup*.
4. Highlight the *Customization Manager* icon. Push the Delete (`Del`) key on your keyboard to remove the icon.
5. Do not use the *Save* or *Cancel* buttons in this step! Close the window using the close `X` in the title bar.

Warning: If you accidentally close the window using a button, fear not, no harm has been done. However, you'll have to repeat the process from the beginning.

6. When prompted, save your changes to the desktop layout.
7. Before closing the *Scripts* panel, highlight the `EXPIP_poplar.py` file, and push the Delete (`Del`) key on your keyboard to delete it - it will be replaced with a new script under a different name.
8. Close all windows and restart Sage.

Once the icon is removed, you're free to upgrade Customization Manager by installing the new version of the module, or selecting the *expip* customization in Customization Manager and upgrading it.

1.8 Uninstall Customization Manager

Customization Manager is distributed as an Extender module. Before removing the application, consider *uninstalling* all the customizations managed by it first.

1.8.1 Uninstall the Module

Uninstall the module from the Extender Modules screen.

1. Open the *Extender –> Setup –> Modules* screen.
2. Highlight the *Customization Manager* module.
3. Press the `Del` (delete) key on your keyboard.

Customization Manager has been removed.

1.8.2 Remove the Icon from the Desktop

If the Customization Manager icon has not been removed from the desktop, you can remove it manually:

1. Open the *Extender –> Setup –> Script* screen.
2. Highlight the first script in the grid, right click, and select *Add to Desktop*.
3. A window with the desktop tree is displayed. Expand *Extender –> Setup*.
4. Highlight the *Customization Manager* icon. Push the Delete (`Del`) key on your keyboard to remove the icon.
5. Do not use the *Save* or *Cancel* buttons in this step! Close the window using the close `X` in the title bar.

Warning: If you accidentally close the window using a button, fear not, no harm has been done. However, you'll have to repeat the process from the beginning.

6. When prompted, save your changes to the desktop layout.

Restart the Sage desktop to complete the removal of the icon.

CHAPTER 2

Packaging Customizations

2.1 Building Packaged Customizations

Customization Manager enables many new development and code deployment workflows. In this article we'll work our way up to a simple workflow that focuses delivered code on customer's business logic, makes code re-use easier, and backporting fixes to existing customers a breeze.

2.1.1 A simple problem

Let's start with a simple use case:

> My customer needs to set an option field to indicate whether an order needs a compliance check before shipping. The field should be set to true if any of the items in the order is in the account set with code "COMP".

Extender makes this easy. One solution is to check orders after update or insert and set the optional field accordingly. At a high level:

```
on insert:
    for each line in the order:
        if the item has an account set code == "COMP":
            set the optional field to True
```

2.1.2 A simple solution

Extender allows us to create a view script that will trigger on Insert or Update of an Order header. Handling the insert event only, we may end up with something like this attached to the OE0520 view:

```
from accpac.py import *

def onOpen():
```

(continues on next page)

(continued from previous page)

```
    return Continue

def onAfterInsert(result):
    """Check the account set of all items and set the optional field."""
    # Open the order lines
    oeordd = self.openDataSource("dsOEORDD")

    # Seek to the first line
    oeordd.browse("")

    # Assume no compliance check is required
    compliance_required = False

    # Check all lines for any item requiring a check
    while oeordd.fetch() == 0:
        if oeordd.get("ACCTSET") == "COMP":
            compliance_required = True
            # If any on item matches we can stop looking
            break

    # Set the optional field, if required.
    # Open the header optional fields
    oeordho = self.openDataSource("dsOEORDHO")

    # Try reading the field to see if it already exists.
    oeordho.recordClear()
    oeordho.put("OPTFIELD", OPTFIELD)
    _read = oeordho.read()

    # If it exists, update the field
    if _read == 0:
        if oeordho.read("VALUE") != compliance_required:
            oeordho.put("VALUE", compliance_required)
            _update = oeordho.update()
            if _update != 0:
                showMessageBox("Compliance Optional Field - "
                               "Failed to set the field.")
    # If it does not, insert a new record
    else:
        oeordho.recordGenerate()
        oroedho.put("OPTFIELD", OPTFIELD)
        oeordho.put("VALUE", compliance_required)
        _insert = oeordho.insert()
        if _insert != 0:
            showMessageBox("Compliance Optional Field - "
                           "Failed to set the field.")
```

Rinse and repeat for the update. This solution isn't pretty and needs a good refactor but it works. With all the logic required to interact with the views readability is not good - it is hard to tease the business logic out.

2.1.3 Another request

Close on the heels of the first request, another customer requires that an optional field be set on an order if an item begins with a particular string.

Patterns begin to emerge, things that are required to deliver on the customer's business logic. For example:

- Open a datasource, clear it, seek to the first record, and iterate. - Check all items in the order.
- Check for the existence of a record using `put` and `read`. - Find whether an optional field already exists.
- Insert or update an optional field. - If it exists, it needs updating, otherwise inserting.

Before starting the second request, let's refactor the reusable parts out from the first one into a new file called `extools.py`

```
from accpac import *

def all_records_in(datasourceid):
    """Generator that yields all records in a datasource."""
    ds = self.openDataSource(datasourceid)
    ds.browse("")

    while ds.fetch() == 0:
        yield ds

def _optional_field_exists_in(datasource, field):
    """Check if a record with field = value exists."""
    datasource.recordClear()
    datasource.put("FIELD", field)
    if datasource.read() == 0:
        return True
    return False

def insert_or_update_optional_field(datasourceid, field, value):
    """Check if an optional field exists, if so update, otherwise insert"""
    ofds = self.openDataSource(datasourceid)
    ofds.recordClear()

    if _optional_field_exists_in(ofds, field):
        ofds.put("VALUE", value)
        if ofds.update() != 0:
            return false
    else
        ofds.recordGenerate()
        ofds.put("FIELD")
        ofds.put("VALUE", value)
        if ofds.insert() != 0:
            return False

    return True
```

Now, let's use our new tools in the solution.

```
from accpac.py import *
from extools import (insert_or_update_optfield, all_records_in, )

CODE = "HAL"
OPTFIELD = "COMPLIANCE"

def onOpen():
    return Continue

def onAfterInsert(result):
    """Check the first characters of items and set the optional field."""
    # Assume no compliance check is required
```

(continues on next page)

(continued from previous page)

```
    compliance_required = False

    # Check all lines for any item requiring a check
    for line in all_records_in("dsOEORDD"):
        if line.get("ITEM").startswith(CODE):
            compliance_required = True

    result = insert_or_update_optional_field(
        "dsOEORDHO", OPTFIELD, compliance_required)

    if not result:
        showMessageBox(
            "Failed to update {} optional field.".format(OPTFIELD))

    return Continue
```

That is a lot more readable - imagine trying to discuss what you've built with a client. Walking through the refactored version is much easier and reads almost exactly like the pseudo-code. The script is focused entirely on the customer's business problem.

2.1.4 Packaging it all up

Now we just need a way to distribute the new package to users along with our script.

Creating a python package is easy, we can create one for our `extools` by creating a new directory in the right format and adding `setup.py` and empty `__init__.py` files.

```
extools/
    |- __init__.py
    |- setup.py
    |- docs/
    |- tests/
    |- extools/
        |- __init__.py
        |- extools.py
```

The special `setup.py` file contains instructions on how to install our package and what it depends on. Our simple package has no dependencies[2] but still requires a simple `setup.py`.

Note that documentation and testing are included directly in the package, along side the code, keeping all the elements of the customization together.

```
from setuptools import setup

setup(
    name='extools',
    version='0.1',
    author='cbinckly',
    url='https://2665093.ca',
    author_email='cbinckly@gmail.com',
    packages=['extools'],
    description='Tools for Orchid Extender.',
    install_requires=[],
)
```

[2] It depends on `accpac.py` but that isn't registered in package indexes.

That is a minimum viable setup file. The `setuptools` package provides loads of other options that include advanced metadata, file installation, shortcuts, and more. Check out the Hitchhiker's Guide to Python Packaging for excellent docs on the topic.

Now that we have our package, we just need to make it publicly available. We can publish on pypi.org or create a VCS repository at Github, bitbucket, or any other VCS server.

Now, the customer can deploy the script and install the `extools` package in two clicks using Customization Manager.

2.1.5 A bug report

All is well until a report comes in of a bug: the first script doesn't always check all records. After some debugging, neither of them (or the three sebsequently delivered) always check all records!

It turns out data sources need to be `.recordClear()` before browsing. It is a simple one line change, before running `<datasource>.browse("")` we need to add `<datasource>.recordClear()`.

For the first customer, a new script needs to be issued. For the second and all subsequent customers, if the `extools` package is updated once all they need to do is a two click upgrade. No fiddling with updated files for each customer, the fix is easily backported.

2.1.6 Conclusion

This is just one example of a workflow that is enabled by easy package management for Extender. It helps keep delivered code focused on the business logic, reduces the time to develop by encouraging and making reuse easy, and makes it simple to deploy fixes and backports to existing customers - all of which improve code quality, customer experience, and decrease support engagements.

See what the extools package has become.

Customization Manager enables us to distribute Sage 300 customizations as Python packages. This has a number of advantages, including distribution leveraging standard Python tools, the ability to automatically install and manage dependencies, and keeping everything, including the documentation, code, and test data, together.

Distributing Sage 300 customizations as Python Packages is a new approach. Although the principles and tools will be familiar to Python developers, they may be new to others. Learn more about *Building Packaged Customizations*

Contents

CHAPTER 3

What is the problem?

Extender uses an embedded Python installation that does not include the usual executables or run in a standards compliant shell. This makes it difficult to leverage Python's package ecosystem when using Extender to customize Sage 300.

CHAPTER 4

How does it work?

Customization Manager for Orchid Extender makes it easy to install, uninstall, and retrieve the information for customizations. It works by temporarily creating a fully functioning Python environment and using pip, the standard Python package manager, to manage the Extender Python installation.

CHAPTER 5

How do I get a copy?

`Download Customization Manager.`

API keys are used to keep your customizations private. Contact us if you need a key.

CHAPTER 6

How do I. . .

Have a look at out *How-Tos* for quick guides to working with the interface and getting common tasks done.

CHAPTER 7

What can I use it for?

Installing and removing customizations! This alone unlocks lots of new opportunities for Extender.

7.1 Install Sage 300 Customizations

If you have a deployment key, Customization Manager allows you to install customizations that have been created for you.

With a standard key, you can install any of the included customizations.

To see a list of the customizations that are currently being distrubuted with Customization Manager, have a look at the Customization Catalogue.

With a developer key, you can manage the full underlying Python installation, publish your own customizations on the index, install customizations from VCS, perform automated testing of customizations and much more.

7.2 Leverage libraries

Need to parse an excel file, do calculations with Julian Dates, access a database or web service like AWS? Maybe a little AI to perform dynamic validations?

Simply build a customization for Customization Manager, leverage the libraries, and save yourself from recreating the wheel.

7.3 Create your own

Create and publish your own libraries to keep DRY and maximize code re-use. Customization code that you use frequently across scripts and re-use the code from your pajckage to save time, improve maintainability, and deliver highly readable code focused on your customer's business logic.

This approach makes backporting bug-fixes to existing customers simple. Simply fix the code, publish a new version of the customization, and upgrade the Customization in one click using the Customization Manager! No need to redistribute new module files.

CHAPTER 8

VCS Support

Pip can install directly from a Version Control System. Supported systems include git, subversion, mercurial, and bazaar. This makes development and testing of customizations much easier: install the customization directly from the code repository in editable mode and upgrades will use the latest commit.

Customization Manager depends on pip, and can do all the things that pip can, so check out pip's docs on working with VCS for supported schemes and syntax.

Note that to download customizations using a VCS the binaries or supporting libraries for the VCS must be installed seperately. At present, the SSH transport protocol is not supported.

CHAPTER 9

Debugging and Viewing the Log

Some customizations cannot be installed. In particular, those that require custom extensions be compiled in place are not supported.

To figure out why a particular action failed, or just to see a transcript of the customization actions performed, use the 'View Log' button. This will open the logs for the last two operations in the default viewer for text (.txt) files.

CHAPTER 10

Help!

If ever things aren't working as expected, a customization you need won't install, or you accidentally uninstall pip (please, never do this), drop us a line and we will help to get you running.