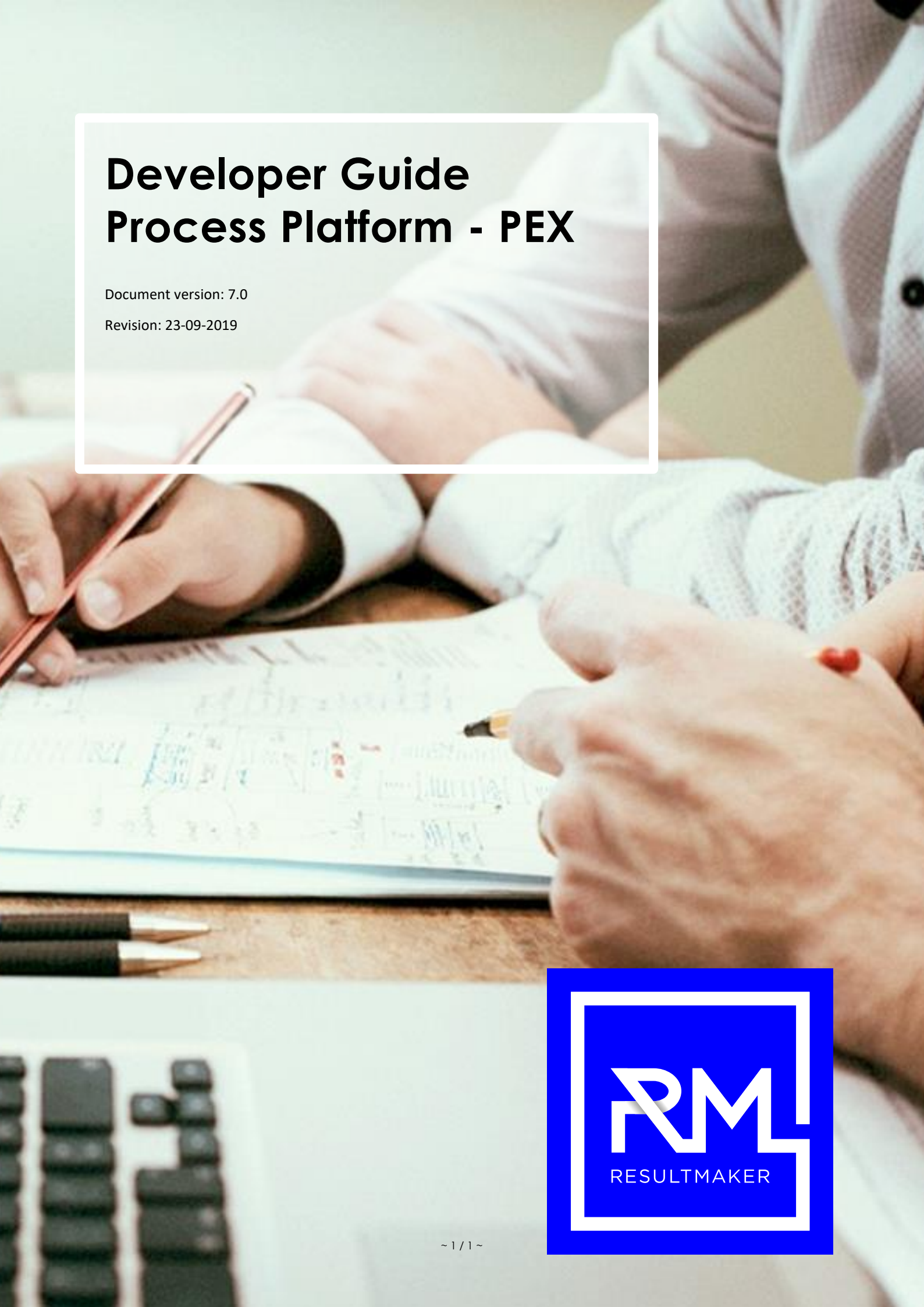

Developer Guide
Process Platform - PEX

Document version: 7.0

Revision: 23-09-2019

RESULTMAKER

PEX

Command-line interface for Process Platform content management. Pex mirrors Process Platform content items as files in a repository. Inspired by the version control system git, content items may be synchronized with a process platform instance by pushing from the repository to the instance, or by pulling from the instance to the repository. Pex is built to work seamlessly with git, so the same folder can serve as a repository for both pex and git at the same time, bringing version control to pex. In addition, pex provides backwards compatibility features such as zip repositories which accept the classic Export tool/Import Service content format and development aids such as the ability to create a visual studio project for a pex repository.

Pex brings the following advantages not previously enjoyed by Process Platform workers:

- Portability of content

While content could already be ported between Process Portal instances it required the use of two different tools with non-obvious feature sets. For example, it was not possible in the export tool to export all content in an instance without providing a list of all content names and the tool could not provide such a list.

- Uniformity of content

All content now has a text file representation allowing anyone to view it without specialized tools. This also makes it easy to search across content types, something that was not possible before.

- Free choice of tools.

While it may well be advantageous in some cases (and strongly recommended for workflows and forms), it is no longer necessary to use specific tools to view or edit files. Any text editor can be used with little friction. This is especially true for export definitions and script files which were always non-GUI files but required special tools to view and edit.

- Easy version control

Being file-based, pex may be combined with any version control system (though it is particularly sympathetic to git) to provide history and rollback features.

Pex repositories

Pex supports two types of repositories; zip repositories and disk repositories. A more accurate name for disk repository might be "file repository" but that name is already taken in company nomenclature.

Zip repository

A zip repository is a zip file containing compressed files each of which represents a content item. A content item is one of six types:

Type	File extension	Description
Project	.ocp	Project/Workflow template xml serialization as rendered by the process engine. Not easily human readable
Questionnaire	.ocq	Questionnaire/Form xml serialization.
Script	.cs	C# process engine script in an xml wrapper.
Formdata	.ocf	Form data/export definition as a Cdata section within an xml wrapper.
SystemParameter	.ocspx	System parameter xml serialization.
File	*	File repository files. Base64 data in an xml metadata wrapper.

The "file" type is special in that it can have any file extension. Any file not of the one of the other five extensions is considered of type "file" and belongs in the file repository. The same goes for any file in a subfolder, even if that file is one of the other five extensions. For this reason, this kind of repository does not support files in the file repository root very well.

The xml-based format used by the zip repository was made purely for transportation purposes and is not well suited for human reading or editing, but it does provide backwards compatibility in that pex can create repositories which the Import Service can process, and pex can read zip files generated by the export tool.

Disk repository

The disk repository evolved from the zip repository and differs in intent. The intent of the disk repository is not just to provide a packaged bundle of content for easy transportation but to make each piece of content readable and, to an extent, editable by human users. To this end, content is "prettified"; xml is indented, cdata sections are unwrapped, base64 is decoded. As well, redundant information is removed, so that for instance if a content item's name appears in the file name, that name will not also appear in the file itself. Thus, it is enough to change the file's name to change the name of the content item, and there is no ambiguity if file name and name in the file content differs.

The disk repository is a directory not a zip file, but otherwise the structure is the same as for a zip repository. The file names also differ slightly from their zip counterparts:

Type	File name	Description

Project	.ocp	A workflow in indented Process Designer xml format minus name entry. Much more readable than the engine format used in the zip repository but care should still be taken not to corrupt the file if editing by hand.
Questionnaire	.ocq	A form in normalized engine format. This means that cdata sections containing xml are unwrapped and indented. This is still somewhat hard to read because of the verbose almost-html and the javascript but it's a big step up from the raw format found in zip repositories.
Script	.cs	The pure script file with no xml wrapper. Will parse in a c# compliant editor.
Formdata	..ocf	The pure formdata xml file without metadata wrapper. Indented for readability and to facilitate change tracking.
SystemParameter	.ocspx	Indented xml with redundant name removed.
File	.	Pure file without metadata wrapper. The file can be edited in any editor suitable for the file type.

Delivery

Pex is delivered as a windows installer and can be downloaded from https://app.resultmaker.com/7/Pex/

The current recommendation is to create a directory named "instances" somewhere. Each instance should then have a directory under the "instances" parent directory named after it. So if you have an instance named "myinstance" and one named "otherinstance" your directory structure would look like this:

```
|
+-Instances
      |
      +-myinstance
      |
      +-otherinstance
```

We'll assume for example purposes that you have the following directory structure right in the c: drive (of the same machine as the instance is running on, unless otherwise specified), but you can put the instances folder anywhere you want, and indeed you can organize the binaries and disk repositories any way you see fit.

Putting pex on the path

Pex is delivered as an installer which puts it on the path of the machine it is installed on.

Version compatability

We are making every effort to have a version of pex cover as many instance versions as possible, and we will do guaranteed backwards compatibility through the content serializations. That is, any version of pex will be able to handle content generated by an older version of pex, but may not be able to connect to an older instance. The only requirement is that the pex reference on the path is left unchanged. In addition, every major version release of pex from 7.0 onwards will ship with the

previous version and a detection mechanism will determine the best version to use. This rule may be extended so that non-major versions are also included, if necessary and as compatibility problems are discovered.

Getting started

After installation you are now ready to run pex:

```
c:\>pex
```

Should yield

```
Error: No command given.
```

Usage

To invoke pex, use the following syntax:

```
pex  [args]
```

Where is one of the following:

Instance-related commands. These interact with a process platform instance:

- List: Lists content on an instance
- Pull: Pulls content from an instance to the pex repository
- Push: Pushes content from the pex repository to an instance
- Init: Initializes a pex repository in the current folder
- Compile: Compiles scripts on an instance
- Status: Shows what instance the repository is connected to and the status of the connection.
- Restart: Restarts the engine of the instance.
- Clone: Clones (copies) content items.
- Get: Gets process engine objects (object instances as opposed to content items).
- Info: Gets engine statistics.
- Poke: Send native engine commands to the instance.
- RemoteVersion: Gets the version of the instance.

Repository commands. These commands interact only with the repository

- Devinit: Sets up various development friendly files such as a .gitignore file and a visual studio project file.
- Listen: Allows pex to mimic the content related endpoints of an actual instance. This means the process designer and similar tools can be used to work directly on the files in the repository.

Other commands. These commands do not directly interact with repositories or instances.

- Gen: Generates native engine commands.
- Version: Gets the pex version.

Instance arguments

All instance commands accept the following arguments:

Argument	Description
[-s | -server <instancename>]	The name of the instance (traditionally called "server", when there was only one instance per physical server). Ex: **-ty localhost**, **-ty pp.resultmaker.com/myinstance**. Default is **localhost.**
[-r | -resource <resource>]	The "resource" part of the endpoint at contains the soap service which can supply instance information. Default is **oc4/accessconfiguration/accessconfiguration.asmx**. If you specify only part of the resource pex will try to guess the rest. So "pe" becomes **pe/accessconfiguration/accessconfiguration.asmx** and "oc/accessconfiguration" becomes **oc4/accessconfiguration/accessconfiguration.asmx**. Usually it will only be necessary to specify the first part. Ex: **-r oc**, **-r processengine**.
[-t | -transport <transport>]	The transport that will be used to connect to the instance: **http** or **https**. May be overridden by the environment.xml instance information file. Default is **http**.
[-p | -port <port>]	The port that will be used to connect to the instance. Default is **80**. Ex: **-p 80**, **-p 443**.
[-u | -user <username>]	The user that will be used to connect to the instance. If none are specified the connection will be made with credentials chosen by the operating system. This will usually be the current user, but may be different depending on the target physical server.
[-pw| -password <password>]	The password which will be used to connect to the instance. This argument is ignored if user is not specified. If the argument is not specified an interactive prompt will appear. Be careful when using this setting, particularly in a script.

Common arguments

All commands accept the following arguments, although for some commands they may have no effect:

Argument	Description
[-bare]	Makes the command output in "bare" or machine-friendly format. Ex: **-ba**, **-bare**. Default is **off**
[-v | -verbose]	Makes the command output extra information helpful for monitoring progress or troubleshooting. The exact nature of the information varies with the command. Ex: **-v**, **-verbose**. Default is **off**
[-frv | -force-remote-version <desired target version>]	Force pex to load use a version of the command suited to the server version given instead of querying the server for its version number. Ex: **-frv**, **-force-remote-version**.

Init Command

The init command creates a pex disk repository in the current directory and links it to an instance so that instance commands may be given without specifying instance information in parameters. The init command takes any of the instance arguments given above and, if an instance is successfully connected to, those arguments will be saved and used as default for all subsequent instance commands run from that directory. It does this by creating a hidden .pex directory under the directory it is run from, so that this information is retained whenever the directory is moved or

copied. Also, deleting this directory will remove the link to the instance. The init command may be run any number of times to link the repository to a different instance.

Examples:

```
c:\>pex init -server localhost/myinstance
```

Creates a pex disk repository in the current directory with a link to the instance localhost/myinstance

```
c:\>pex init
```

Creates a pex disk repository in the current directory with a link to the default instance (localhost)

```
c:\>pex init -resource pe
```

Creates a pex disk repository in the current directory with a link to the default instance (localhost) but with the process engine endpoint under "pe" instead of the default "oc4".

List Command

The list command lists content on the server by name without pulling it from the repository. It is the simplest query command, and some of its arguments apply to the other query commands as well (eg. Pull).

Query arguments:

Argument	Description
[-ty | -type | -types <types>]	Restricts the query to the specified content types separated by commas. Defaults to **Project**,**Questionnaire**,**Script**,**Formdata**,**SystemParameter**,**File**. EX: **-ty project**, **-ty script, file**
[-a | -analyze | -re | -recursive]	Flag. If this is enabled, in addition to the content found, all content required by that content will also be added to the result of the query. Ex: **-a** (on), **-a-** (off). Default is **off**.
[<filter>]	A content name filter which will be applied to the query results. Leading and trailing asterisk (*****) counts as wildcard. To get all content beginning with the string "softwaretest" use **softwaretest***. The match is not case sensitive. Default is *****.
[-pr | -prettify]	Flag. If this is enabled, pex makes an effort to present the content in a human-readable and version control friendly way. See the "Disk repository" section. If it is off, the content will be pulled in the classic serialization format which is compatible with the import service. This flag has no effect if the -zip flag is on; zip repositories will always use the unmodified serialization format. Ex: **-pr** (on), **-pr-** (off). Default is **on** for disk repositories.
[-wh | -whitelist]	Flag. If this is enabled, pex uses a whitelist of file paths to filter what it finds. The whitelist is taken from stdin. This enables easy interfacing with other command line tools. For example, this may be used to make a repository solution-specific by keeping the file names of the files belonging to the solution in a text file and piping it to the pull and push commands. Ex **-whitelist** (on), **-whitelist-** (off). Default is **off** unless pex detects that stdin is redirected, in which case it is **on**. Cannot be turned on when pushing zip repositories.
[-tfn | -typed-formdata-	Flag. If this is enabled, pex includes the formdata type names in names of formdata content. This is a legacy behavior of pex and should only be used to convert legacy pex repositories to the new standard. Ex **-typed-formdata-names**

names]	(on), **-typed-formdata-names-** (off). Default is **off**. Does no apply to zip repositories.
[-lt | -listtypes]	Flag. If this is enabled, instead of listing content the command will list all content types that the instance understands. This is handy to get the exact spelling. Ex **-lt** (on) **-lt-** (off). Default is **off**.
[-ser | -select-referencing]	Flag. Instead of emitting the the content items found, emits all the content items that **reference** one or more of those items. This lets you easily find, say, all the projects that reference a particular form. This command is quite slow as it relies on analysis of the full set of content items on an instance. It may therefore be preferable to run it using the -local parameter. Ex **-ser** (on) **-ser-** (off). Default is **off**.

List arguments:

Argument	Description
[-loc | -local]	Flag. Runs to command on the local disk repository rather than a remote instance. For time consuming analysis it is often better to pull all content of an instance to a local repository and work on it there and this parameter facilitates this process. The local run is done through the 'Listen' command and the restrictions that apply to that command apply here too. Ex **-loc** (on) **-loc-** (off). Default is **off**.

Pull Command

The pull command pulls content from an instance to the repository. It takes the same query arguments as the list commands takes, so the list command can be used to make a "dry run" for the pull command. In addition, the pull command takes the following arguments:

Argument	Description
[-z | -zip]	Flag. If this is enabled, pulls to a zip repository rather than a disk repository. Ex: **-zip** (on), -zip- (off). Default is **off**.
[-bl | -blacklist]	Flag. Prevents certain specific content items from being imported. These are the scripts and forms that enable process designer functionality and the environment.xml file which contains instance information used by pex and other tools. Ex **-bl** (on), **-bl-** (off). Default is **on** for both disk and zip repositories. Note that this is different for the Push command.
[-o | -outfile <outfile>]	If the zip flag is enabled, this argument can be used to specify a file name for the zip file. It has no effect if the zip flag is off. Ex: **-outfile somecontent.zip**. Default is a **generated**, **timestamped** file name.
[-b | -backup]	Reserved for future use.

A disk repository pull always pulls to the current directory.

Push Command

The push command pushes content from a repository to an instance. It takes the following arguments:

Argument	Description
[-z | -zip]	Flag. If this is enabled, pushes from zip repositories rather than the current disk repository. Ex: **-zip** (on), **-zip-** (off). Default is **off**. .

[<filter>]	Filters content to push. Ex: *.ocp. The behavior of this arugment differs depending on whether or not the "zip" flag is specified. If specified, each and all files selected by the filter will be treated as a zip repository, and all content therein will be pushed. If the zip flag is not specified, all files selected by the filter will be pushed. If this agrument is not specified, and the zip flag is enabled, pushes all zip repositories in the current directory. If zip is not enabled, ordinary content types are pushed if they are in the root directory and "File" content type files are pushed only if they are in the "private" or "public" subfolder or recursively in a subfolder of one of those folders.
[-pr | -prettify]	Flag. As with the pull command, specifies whether the content is in the raw, legacy format or the new, prettified format. A good candidate for autodetection in the future. Ex: **-pr** (on), **-pr-** (off). Default is **on** for disk repositories, always **off** for zip repositories.
[-ba | -backup]	Flag. Creates a backup folder and pulls content about to be replaced on the instance to this folder before pushing. Ex **-ba** (on) **-ba-** (off). Default is **off** for disk repositories, **on** for zip repositories.
[-bl | -blacklist]	Flag. Prevents certain specific content items from being imported. These are the scripts and forms that enable process designer functionality and the environment.xml file which contains instance information used by pex and other tools. Ex **-bl** (on), **-bl-** (off). Unlike the pull command, default is **on** for disk repositories and **off** for zip repositories. This is similar to the behavior of the traditional system and ensures that pex push behaves just as import via the import service does.
[-wh | -whitelist]	Flag. If this is enabled, pex uses a whitelist of file paths to determine what to push. The whitelist is taken from stdin. This enables easy interfacing with other command line tools. For example, this may be used in conjunction with git to push only those files which have changed in the last commit or only those files which differ from those in a different repository. Ex **-whitefish** (on), **-whitelist-** (off). Default is **off** unless pex detects that stdin is redirected, in which case it is **on**. Cannot be turned on when pushing zip repositories. Turning the whitelist off explicitly may be necessary when pex detects that input is redirected but this is for reasons other than providing a whitelist. This happens on remote deployment via teamcity for example.
[-lo | -log]	Flag. Creates a log of the push to file. Always on for zip repositories.
[-tfn | -typed-formdata-names]	Flag. If this is enabled, pex includes the formdata type names in names of formdata content. This is a legacy behavior of pex and should only be used to convert legacy pex repositories to the new standard. Ex **-typed-formdata-names** (on), **-typed-formdata-names-** (off). Default is **off**. Does no apply to zip repositories.
[-nc | -nocleanup]	Flag. Disables 'clean up' of original source zip file when using zip import. The default behavior is to move the zip file to a backup folder to signify that the import is done. Ex **-nocleanup** (on), **-nocleanup-** (off). Default is **off**.

DevInit Command

The devinit command adds some files to a directory to aid in using pex with development and administration tools. First, it adds a .gitignore file which ignores the same files as pex does. This allows a git repository in the directory to easily keep track of pex content. It also creates a visual studio solution and project file containing all the .cs files in the directory. This enables easier editing and intellisense support in tools such as Visual Studio and Visual Studio Code. This

command may be run any number of times and will overwrite the files it generates. For that reason it is not recommended to make manual changes to these files.

The command takes the following arguments.

Argument	Description
[-git]	Flag. If this is enabled, in addition to the usual effects, the command initializes a git repository in the folder. Ex: **-git** (on), **-git-** (off). Default is **off**.
[-vstoolsversion | vstools <visual studio tools version>]	Chooses the visual studio tools version to stamp in the project file that is created. Ex: **-vstools 12.0**. Default is **14.0**.

Compile Command

The compile command compiles scripts on the target instance. It will show whether compilation failed or succeeded and also if the instance needs to be restarted for succesful compilation to take effect. This is the case with some scripts if they are shared as includes among multiple scripts.

Except for the -type and -ser parameters, which have no effect, it takes the same query arguments as the list commands takes, so the list command can be used to make a "dry run" for the compile command.

Status Command

The status command shows information about a pex repository. It will show if a directory is "linked" to an instance, meaning if the pex init command has been run in the directory and if so what instance it is linked to. It will also show if an http connection can currently be made to any linked instance.

The command takes no arguments.

Restart Command

The restart command restarts the target instance. An instance may need to be restarted to allow it to reload newly compiled scripts or to resolve abnormal behavior. Be careful when using the restart command in a production environment, as users may be adversely affected.

The command takes the following arguments.

Argument	Description
[-ns | nostart]	Flag. Skips starting the instance after stopping it. Ex: **-ns** (on), **-ns-** (off). Default is **off**.

Listen Command

The listen command allows pex to function as a Process Platform instance for content. This allows the Process Designer and other tools (such as the visual studio script extension and pex itself) to load and and save to a pex repository instead of a live server.

The command is under development and is feature complete for use with the Process Designer but it does not support all functionality of other tools (such as pex itself) yet.

Important!

When using this command on a machine on which Skype is installed, make sure Skype is not running. Otherwise the command will fail with the message:

```
Error: The process cannot access the file because it is being used by ano
ther process
```

The command takes the following arguments:

Argument	Description
[<route>]	Instance url of the emulated instance. Ex: **http://localhost/pex/**. If the url does not contain a trailing '/', one will be added. Be careful using http://localhost/ as this may interfere with IIS on your machine. Also, while 'localhost' will work for all connections even if the url uses the server name or ip the reverse is not true, that is, if you set up the listen endpoint on http://myserver/pex you cannot reach it form the process designer at the url http://localhost/pex even if 'myserver' is the name of the local machine that you are running both pex listen and the process designer on.
[-bl | -blacklist]	Flag. Prevents certain specific content items from being imported. These are the scripts and forms that enable process designer functionality and the environment.xml file which contains instance information used by pex and other tools. Ex **-bl** (on), **-bl-** (off). Default is **on**. Since the process designer generates standard, blacklisted content if it finds it to be missing, this flag is useful if working on a pex repository which is intended to be free of blacklisted content.

Clone Command

The clone command makes a copy, or clone, of one or more content items.

A cloned item will get a new id (name) based on the original id but with a version suffix, so that **MyContentItem** becomes **MyContentItem_v1**. However if a content item is already versioned the clone will get an identical name with an incremented version, so that **MyContentItem_v1** becomes **MyContentItem_v2**.

A clone can either be a shallow clone, where only the content item itself is copied while references to other content items are left alone, or a deep clone where referenced items are also cloned. Or it can be a combination of the two. We control this by selecting all the content items to clone as a set using the normal query capabilities and whitelist capabilities of pex. All content items in the set will be cloned, and each reference to a member of the set will be updated to be a reference to the clone. References outside the set will remain untouched. This means we can fine control what items are deep cloned, so that we may avoid cloning library scripts, for example, while still easily supporting the main usage scenario: a deep clone of a workflow.

While this is the general gist, the clone command does come with some limitations. First, it only supports cloning workflows, forms and scripts. Support for other types such as formdata, file repository files and server parameters may come in the future, but until then, this limitation must be worked around. This is not as bad as it sounds. For most practical purposes, formdata are generated by workflows (using the process designer) and so all the information necessary to generate them are contained in the workflows. As part of the workflow cloning process, new integer ids are generated for transactions and signing steps which emit formdata. By saving the workflow in the process designer, new formdata will be generated exactly as if it had been cloned. This can also be accomplished directly using the clone command with the **generate-formdata** argument. File repository files will not be cloned though, and so must be handled manually.

The command takes all It takes the same query arguments as the list commands takes, in order to specify the set of content items to be cloned. In addition, it takes the following arguments:

Argument	Description
[-gf | -generate-formdata]	Flag. Instructs pex to generate formdata for each cloned workflow. This is typically used when making a deep clone of a workflow.
[-ni | -no-increment]	Flag. Prevents pex from incrementing versions when generting new names. **MyContentItem_v3** becomes **MyContentItem_v3_v1**.

Examples:

```
c:\>pex clone MyWorkflow.ocp -s localhost/myinstance -a -gf
```

Clones the *'MyWorkflow* workflow on PE instance **localhost/myinstance** and all qualified content items recursively referenced by that workflow. References in clones are updated to reference clones as appropriate. Finally generates form data for the workflow clone.

Gen Command

Generates a process engine command wrapped in a command list which may be sent to a process engine instance with the poke command. Commands may be chained so that several are contained in the same command list. A rudimentary backreference notation enables output from one command to be used as input to another. The following process engine commands are supported:

- AcceptInvitation
- AddUserToRole
- AutoCloseActivity
- AutoCloseAll
- DeleteProject
- GetActivityIdFromPath
- GetInterviewInfo
- GetInterviewPage
- GetItemDetails
- GetNextActivity
- GetProjectList
- GetProjectList2
- GetProjectStructure5
- GetProjectVariable
- GetProjectVariables
- ResetItem
- SetInterviewAnswers
- SetVariable
- StartItem
- StartProject
- TakeRolesFromConsentingUser

In addition there is the command Commandlist which generates no cammand but only the command list wrapper. For further info on the individual commands see Pex Gen Commands.

It takes the following arguments:

Argument	Description
[<subcommand>]	The name of the process engine command to generate.
[<-st | -security-token>]	The security token used to authenticate to the process engine. Default is the windows security token **WinDelegation:OUTOFBANDPAYLOAD==** which is user independent and merely tells the engine to get authentication information from context instead of the token. For most purposes you should leave this alone.
[<-parameters>]	Process engine context parameters. An series of key-value pairs in format key=value separated by white space.

Example commands and outputs:

```
c:\>pex gen getprojectstructure5 --wf 1


    1


c:\>
```

This creates a command for getting 'project structure 5', the workflow data that the front end uses to generate the workflow menu for the workflow instance with workflow id 1.

```
c:\>pex gen getnextactivity --wf 69 | pex gen getinterviewinfo --wf 69 --
ac %command(0).returnvalue(0).value%


    69


    69
    %command(0).returnvalue(0).value%


c:\>
```

This chains two commands for getting the interview info (some interview information required by the front end) from the 'next activity' on the workflow instance with id 69.

```
c:\>pex gen commandlist --
parameters "UserAgentUrl=http://myserver/fe/default.aspx?AddRoles=Role2&RemoveRoles
=Role1" | pex gen getnextactivity --wf 69 | pex gen getinterviewinfo --
wf 69 --ac %command(0).returnvalue(0).value%
```

http://myserver/fe/default.aspx?AddRoles=Role2&RemoveRoles=Role1

```
69
```

```
69
%command(0).returnvalue(0).value%
```

```
c:\>
```

As above, but a UserAgentUrl context parameter is sent with the commands with a query that simulates that we remove ourselves from role1 of the workflow and add ourselves to role2 for this command batch only. This only works if executed by an instance administrator (ordinary users cannot do this), and is a useful way to see a workflow from a user's point of view.

Get Command

Gets process engine objects and emits them to stdout. The command currently only supports getting workflow instances but more types (files in the file store, users) may be added later.

It takes the following arguments:

Argument	Description
[<object identifier>]	An id uniquely identifying the object to get.

Example:

```
c:\>pex get project/1234
```

Gets the workflow instance with workflow instance id 1234

Info Command

Gets statistics from the process engine. This feature is currently under development.

Poke Command

Sends a process engine command list to a process engine instance.

It takes only the standard pex instance command parameters. The command to be sent is provided via stdin.

Example:

```
c:\>pex gen getprojectstructure5 --wf 69 | pex poke
```

Sends a command list generated by pex gen to the linked instance.

```
c:\type asavedcommandlist.xml | pex poke
```

Sends a command list contained in a file to the linked instance.

RemoteVersion Command

Gets the version of the linked instance.

Version Command

Gets the pex version.

Tips & Tricks

Pex is a powerful tool on its own but really shines when combined other batch commands and command line tools. Here are a few examples:

Find Dependencies

Pex does not currently support finding the content items that depend on a particular content item. However, by harnessing the analyze (or recursive) functionality which does the opposite, that is, finding the content items that a particular item depends on, a cartesian product of content items and their dependencies may be constructed:

```
c:\>pex list --
bare > cand.pex && for /F "tokens=*" %A in (cand.pex) do (pex list "%A" -
a -
bare > temp.pex && for /F "tokens=*" %B in (temp.pex) do echo %A,%B >> re
sult.txt)

or

c:\>echo > result.txt && pex list --
bare > cand.pex && for /F "tokens=*" %A in (cand.pex) do (pex list "%A" -
a -
bare > temp.pex && for /F "tokens=*" %B in (temp.pex) do echo %A,%B >> re
sult.txt)

to make sure the output file is cleared before we start building it
```

This takes a **long** time to run, but after it's done, the main output file (result.txt) may be searched for dependencies. You can use a text editor or excel, or, on the command line, if we were to search for, say, items that depend on the form **SoftwareTest_AllFormElementTypes.ocq**:

```
c:\>find "SoftwareTest_AllFormElementTypes.ocq" result.txt
```

Which might yield:

```
---------- RESULT.TXT
SoftwareTest_AllFormElementTypes.ocq,public\static\rm\graphics\RMLogo.jpg
SoftwareTest_AllFormElementTypes.ocq,SoftwareTest_AllFormElementTypes.ocq
SoftwareTest_BackendValidation.ocp,SoftwareTest_AllFormElementTypes.ocq
SoftwareTest_HtmlMergeReceipt.ocp,SoftwareTest_AllFormElementTypes.ocq
SoftwareTest_PE.5.2_01_Prefill.ocp,SoftwareTest_AllFormElementTypes.ocq
```

Items before the comma depend on items after. In this example the workflows **SoftwareTest_BackendValidation.ocp**, **SoftwareTest_HtmlMergeReceipt.ocp** and

SoftwareTest_PE.5.2_01_Prefill.ocp depend on the form. It also depends on itself by definition and in turn depends on **public\static\rm\graphics\RMLogo.jpg**. That last piece of infomation, though, we could have easily gotten just by doing:

```
c:\>pex list SoftwareTest_AllFormElementTypes.ocq -a
```

Alternate zip format

The zip format precedes the disk repo format and exists for reasons of backward compatibility. However, a zip format which is human readable like the disk repository files yet convenient for transport and comparmentalization like the zip format is superiour for most purposes. Pex does not support compressed disk repositories natively, but chaining commands can achieve the same effect. The command:

```
md tzpex && "c:\program files\7-zip\7z.exe" x z.zip -
otzpex && cd tzpex && pex push -
s localhost/x && cd .. && rmdir tzpex /S /Q
```

Will create a temporary directory 'tzpex', extract the z.zip file to the temp directory and push from that directory to the instance at http://localhost/x. It will then delete the temp directory. This is assuming that z-zip is installed in c:\program files\7-zip. If 7-zip is on the path, as it is if your version of pex is new enough, you can simplify to:

```
md tzpex && 7z x z.zip -otzpex && cd tzpex && pex push -
s localhost/x && cd .. && rmdir tzpex /S /Q
```

A similar technique can be used to pull content to an alternate format zip container.

Version

This document is based on the functionality in Pex 7.0.2703 or higher.