PERFORMANCE.READY Tools

WxConfig

Version 1.8

Contents

Contents

1 Overview

WxConfig is a comprehensive configuration management framework for Software AG's Integration Server that makes solutions agile and deployable into new environments with zero manual activity. It thus helps to reduce manual effort, risks, and costs.

What WxConfig does, in essence, is to ensure that the environment is configured correctly and the application, services, etc. running in it have the right settings available at run–time. This is achieved by selectively loading only those configuration values that are appropriate for any given system. Those values can be stored in files (local or shared storage) or database tables. In addition to simple key–value pairs (also referred to as properties) it is also possible to use XML structures and query them using XPath. All this is supported through a rich web–based UI.

This document describes the functionality of WxConfig. That entails a short look at the design principles, an introductory step–by–step guide, and a reference of services. Also, some of the more advanced topics are covered.

DISCLAIMER: The WxConfig package is not an official Software AG product and as such provided on an as-is basis without any support.

2 Installation

2.1 Requirements

Integration Server

The package has been used successfully with the following versions of Integration Server

- 8.2, 9.0, 9.5, 9.6, 9.7, 9.8, 9.9, 9.10, 9.12, 10.0

Given the nature of its implementation there should be no problems using it with other supported versions of Integration Server. But please also note that the package uses un–supported APIs and therefore it cannot be guaranteed to work with future versions of Integration Server.

Java

Until v1.7.x the package is being maintained on a v8.2 installation and hence Java services are compiled for Java 6. If you are still running on Java 5 and get error messages about missing class files during package startup, please get in touch with Software AG to discuss this.

Starting with v1.7.2 the build process takes place on a v9.12 installation using Java 8. This change was necessary due to Java classes needed for fixing a bug when creating Ehcache caches using auto-setup.

External Dependencies

None

2.2 Package Installation

The WxConfig package is installed like any standard Integration Server; no special steps are required:

- Copy the `WxConfig.zip` archive to `$SAG_HOME/IntegrationServer/instances/<INSTANCE>/replicate/inbound` or `$SAG_HOME/IntegrationServer/replicate/inbound` if your installation does not support mutliple Integration Server instances within a single installation.
- Open the Integration Server Admin Console and go to *Packages / Management*
- Click *Install Inbound Releases* and select `WxConfig.zip` from dropdown list
- Click *Install Release* button
- Refresh the Integration Server Admin page
- Check server log for relevant messages

2.3 License

Since v1.3 WxConfig requires a license, which customers can request through their designated contact from Software AG's Global Consulting Services. This license is version–specific, so a license for v1.5 will not work with v1.6.

The license must be entered in web UI by clicking *Settings / License*, pasting the contents into the editor window, and clicking the *Save* button. This will also trigger a reload of configuration files and could therefore interrupt running transaction. It is therefore recommended only for systems that are idle at the time.

Alternatively, it is possible to simply replace the license file on the file system. WxConfig will check for a license update once a day and reload it silently in the background. This approach will not affect running transactions and is therefore recommended for all non–development systems.

If the license is about to expire, the user will be alerted in the UI as well as the log files (`server.log` and `WxConfig.log`). When the license is valid only for 60 days or less, the messages are issued as warnings. When 30 days or less have been reached, this changes to an error.

3 Getting Started

The purpose of this chapter is to provide a very basic understanding about WxConfig (more details are provided in chapter 4). It starts with a description how WxConfig works and after that things get practical with a step–by–step guide how to use it.

3.1 How It Works

For each package that manages its configuration with WxConfig, the latter holds all defined values in memory. Values for different packages are kept totally separate, so the exact same key can be used in multiple packages without issues. It is also possible to access values from other packages (for details see section 6.2, p. 33).

What WxConfig does at startup is scan all active packages on Integration Server and look into their ./ config/ sub–directories for a file called wxconfig.cnf (the main configuration file). If it is found, the package is assumed to be using WxConfig. This main configuration file is then loaded into memory and parsed for instructions to include (i.e. load) other files. If no inclusions are found, the startup is finished for the given package and all values from the main configuration file can be used to control the behavior of the logic contained in the package.

In reality, though, most packages will use inclusions to meet their requirements. These inclusion instructions can be conditional or unconditional. Unconditional means that the file is loaded on every system. The main reason to have unconditional files in addition to the main file (wxconfig.cnf) is to either split the configuration into separate parts (e.g. one file per functional area) or for XML files. Conditional inclusions mean that the files get loaded only if e.g. the system is designated as the development environment (*DEV*); or if the system is running on Windows as an operating system. Inclusions are only possible from wxconfig.cnf, but not other files that were included from there.

Once it was determined for each package, which files are to be loaded on the current system, WxConfig will do that.[1] The contents of all files will be merged into a single configuration space for each package.[2] This means that when a particular value is retrieved later, it is completely transparent from which file it originated. The consumer of a configuration value does not need to know anything about where the value came from.

Whenever a package was determined to be using WxConfig (by means of checking for wxconfig.cnf in the ./config/ sub–directory), a package dependency will automatically be added.[3] This avoids all sorts of weird and hard–to–diagnose run–time errors.

WxConfig logs a lot of information about its operations. Just a few lines go the standard server.log of Integration Server (incl. version, license information, and the location of the main log file of WxConfig). The main log file for WxConfig sits in the Integration Server's ./logs/ directory and is named WxConfig. log. It is possible to view it from the UI (*Logging / Display Log File*). It is highly recommend to study the log file when starting to use WxConfig. Its contents (esp. if the log levels are changed to Debug) reveal a lot about how WxConfig works.

---

[1] This is a late–binding approach, while e.g. application servers usually use early binding.
[2] In technical terms it is a Java map of objects, each holding a so–called composite configuration.
[3] It is possible, although generally not recommended, to disable this; for details see section 9.6, p. 47

3.2 Your First Package with WxConfig

You will now start creating your first own package that will make use of WxConfig. It entails the following steps:

1. Enabling the custom package for WxConfig, thus automatically creating the main configuration file (`wxconfig.cnf`)

2. Placing values into `wxconfig.cnf`

3. Consume values in Flow using the respective service

3.2.1 Creating the configuration

- Create the package "TestWxConfig" and place a folder with the exact same name into it

- Switch to web GUI of Integration Server, go to *Solutions* menu and click *Config. . .* there
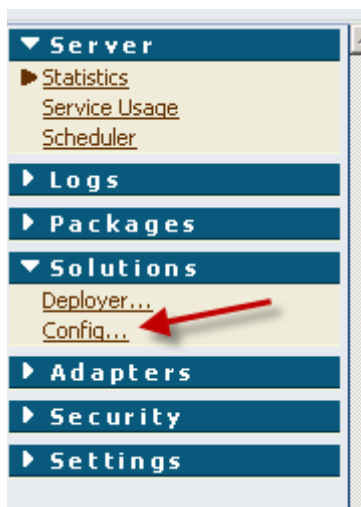
- Enable WxConfig for your new package

If all went well, the following should be seen

- Click the "edit" link to start working with the file[4]

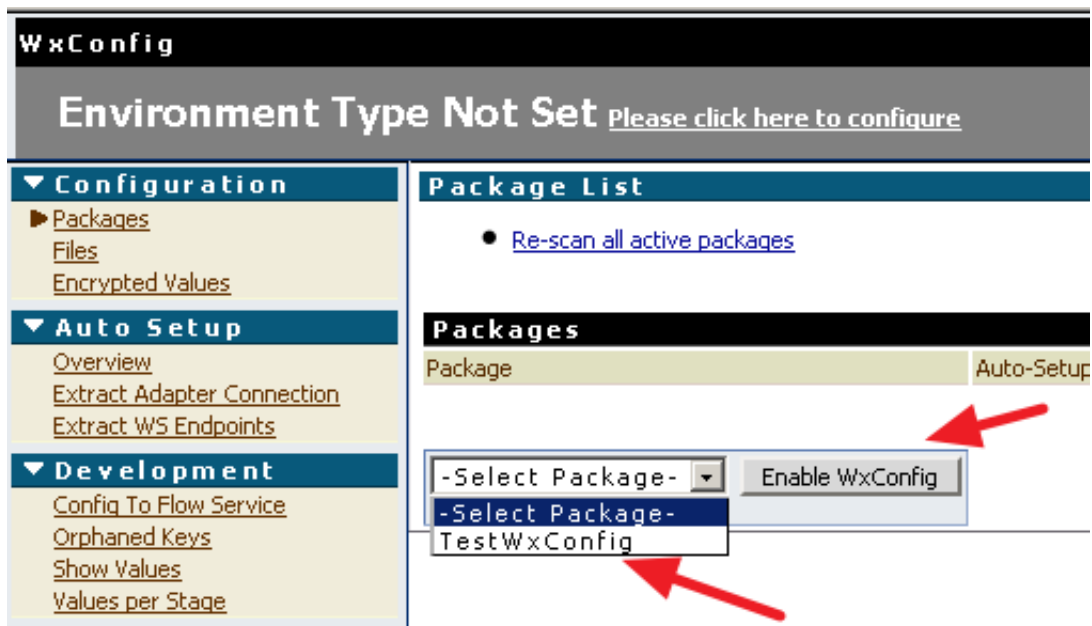Every new file is pre-populated with a template. The content of the templates (properties and XML) is configurable in the menu *Settings / Templates.* You would normally perform changes here to comply with corporate or project guidelines.

- Remove the template content and enter a sample key / value pair and save your configuration

---

[4] The built-in editor used to have issues with some versions of Internet Explorer, which is fixed in **WxConfig** v1.5 and higher.
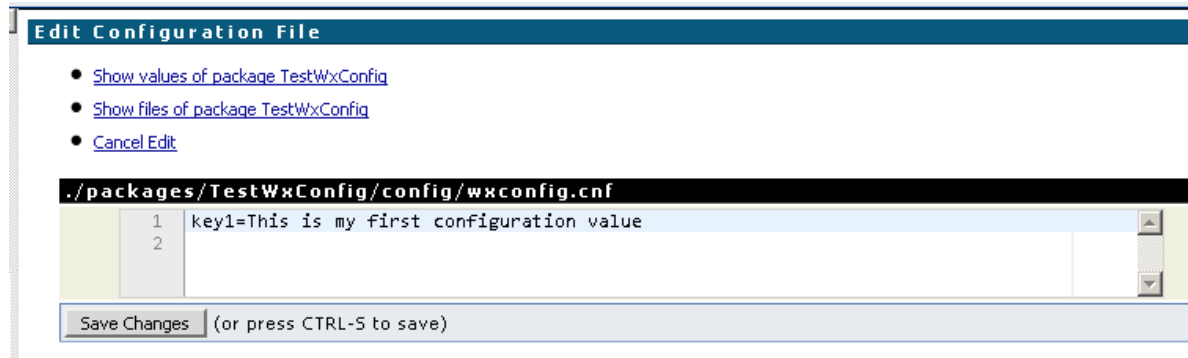
- The result will be as shown below
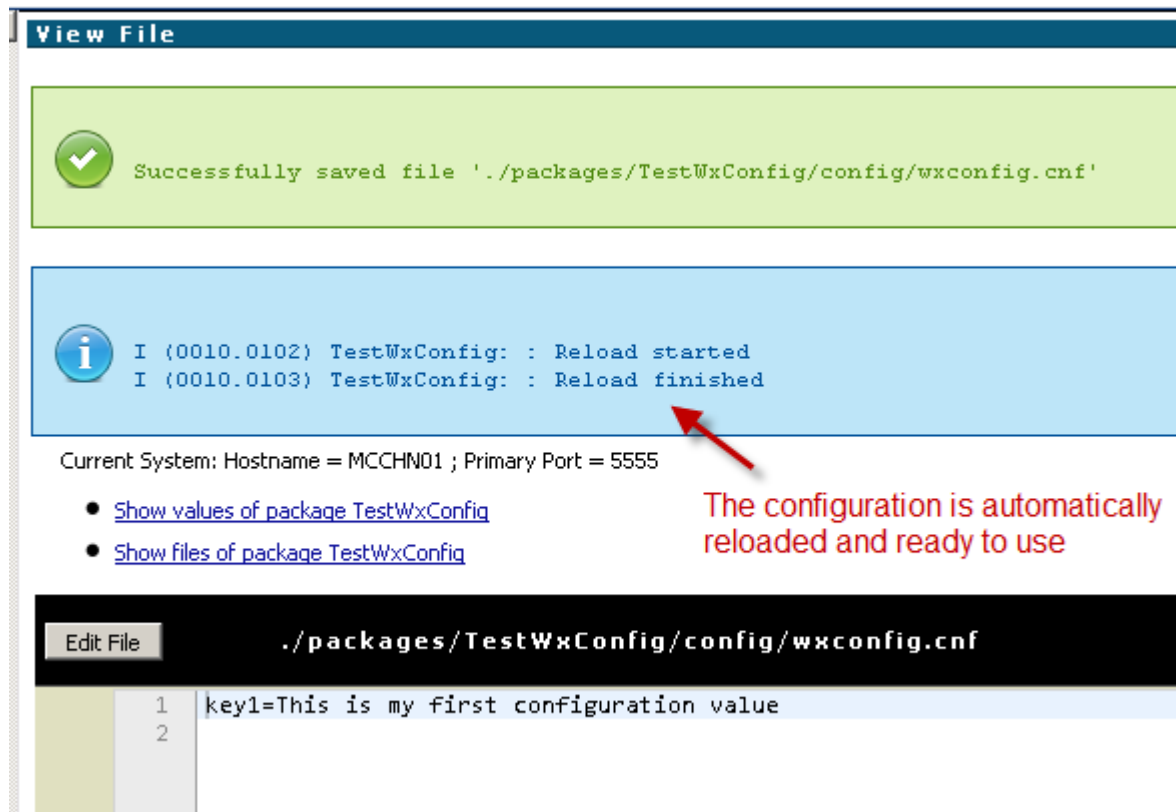
3.2.2 Using the configuration

To test things switch to **Designer**, locate the service `wx.config.pub:getAllValues` delivered with the **WxConfig** package and run the service with the respective input parameters.
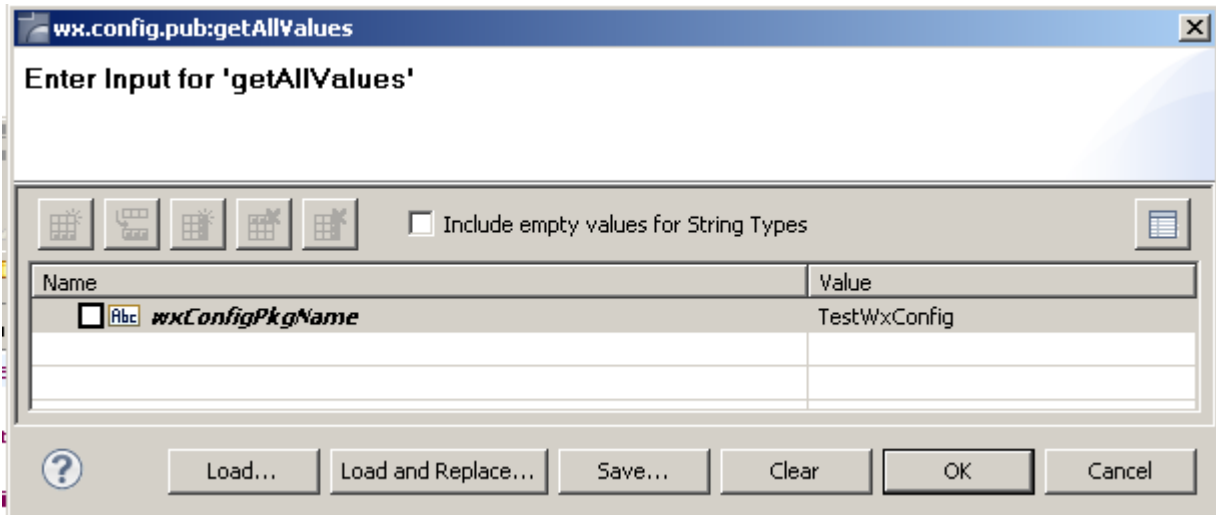
- The result should look like this

  This gives you just a first overview of the content of your configuration file.

- To get things more useful, we now will add the just created configuration value (named `key1`) to a **Flow** service. To do so, create a **Flow** service named `configTest` within the above created folder and copy its location to your clipboard:
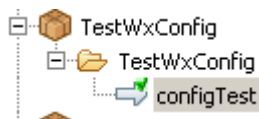
- Switch back to the **WxConfig** console, choose *Development / Config To Flow Service* and paste the location of the newly created Flow service to the field *Service Name*:
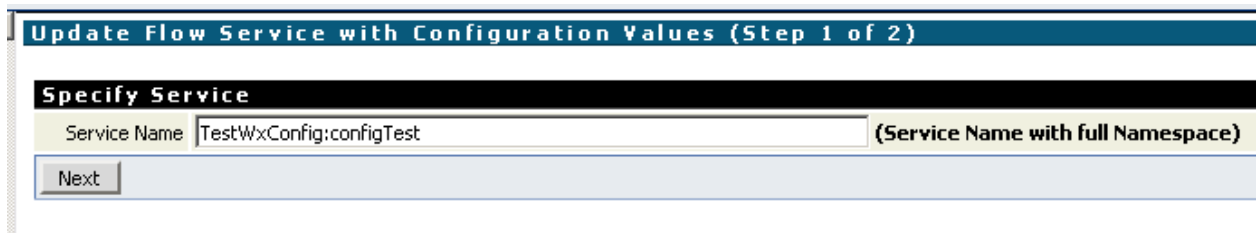
- After clicking the *Next* button, you can configure which of your configuration keys should be added to the selected **Flow** service. In addition you can choose whether values should be added as string or string list (when *Add as List* is checked). Furthermore, you can specify a different variable name for the service, but keep in mind, that it might be hard to remember their link, if their naming differs!

- Click on *Create/Update Service* and switch back to your **Designer**, where you refresh your service and should find the following, generated Flow steps:
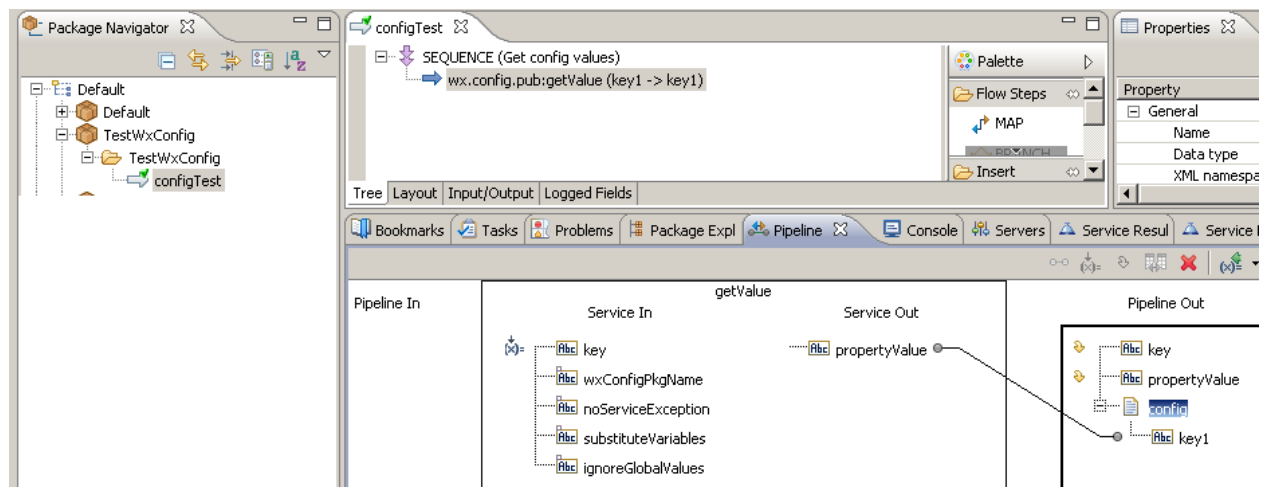
- Run the service and you will receive the following result:

- If you get an error "Could not run 'configTest' com.wm.app.b2b.server.ServiceException: Key 'key1' did not return any value", you can solve this by reloading configuration files.

3.2.3 Getting Rid of Orphaned Keys

As orphaned keys are hard to recognize, WxConfig helps you to figure them out. Therefore the tool scans for services which have a sequence with the comment *Get config Values* as the first flow step within a service. In a second step a check will be done, whether invocations of `getConfigValue` or `getConfigValueList` services exist that use a non–existing configuration values.

To check this feature, do the following steps:

- Open your `wxconfig.cnf` and change the key name to `key1a` and save your changes:

- As the flow service still tries to read the key named as `key1` and this key does not exist any more, the web UI lists this one under *Development / Orphaned Keys*:
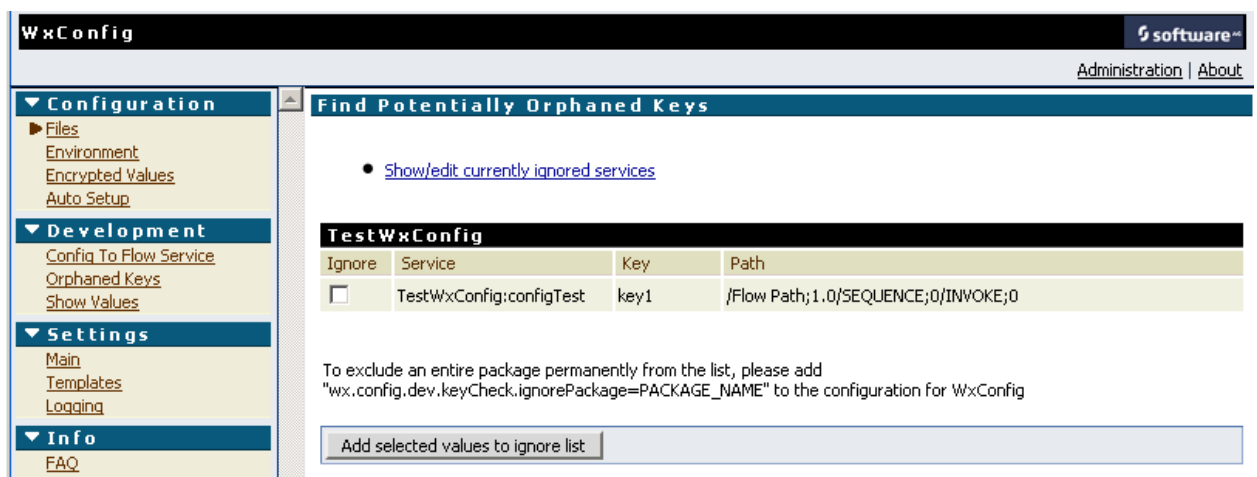
If you do not want the tool to list specific keys as orphaned, you can check them with *Ignore* and *Add selected values to ignore list*. If you want to exclude a whole package from the orphaned list, you can add `wx.config.dev.keyCheck.ignorePackage=PACKAGE_NAME` to the file `wxconfig.cnf` of the WxConfig package.

4 Fundamentals

This chapter gives a more detailed overview of the various parts of WxConfig. It is highly recommended to go over it completely for a good understanding about all the things where WxConfig can help.

4.1 Web GUI

The web GUI consists of the following parts:

- Configuration: Access to packages, files, and encrypted values
- Auto–Setup: Working with environment setup
- Development: Help during the development process
- Logging: Logging and auditing
- Settings: Configure various aspects
- Info: Information and help (incl. access to samples package)
- Plugins: Available plugins (currently only Subversion integration for run–time changes)

4.1.1 Section: Configuration

The package registers itself as a solution and can be accessed directly from the main menu of the Integration Server admin console as shown below.

The start screen shows a list of all packages that are using WxConfig. For a quick overview the main status indicators are also displayed.

It is also possible to create new configuration files directly from the GUI as indicated below.
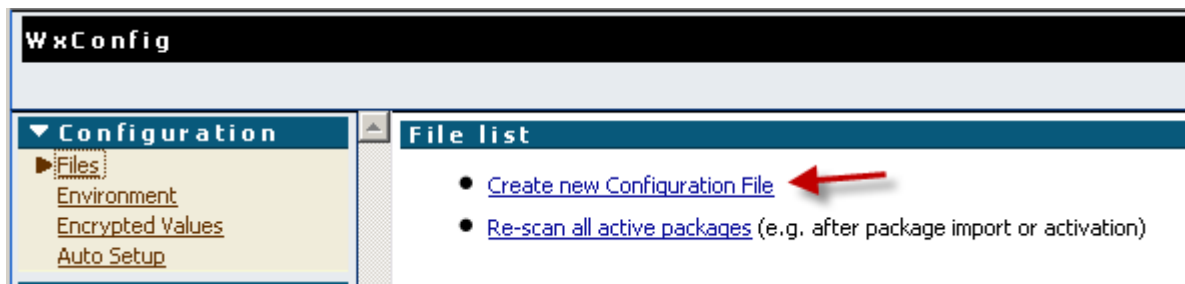
You can have your new configuration file pre-populated with the content of an existing one. This is particularly useful when creating them for several environments, where the same parameters (but with different values) must exist in all of them.
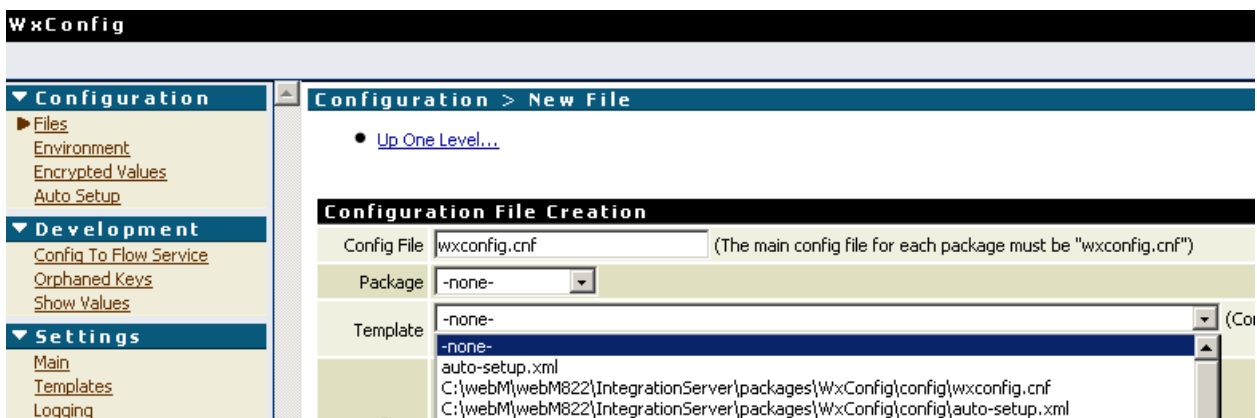
You can directly edit files in the web GUI
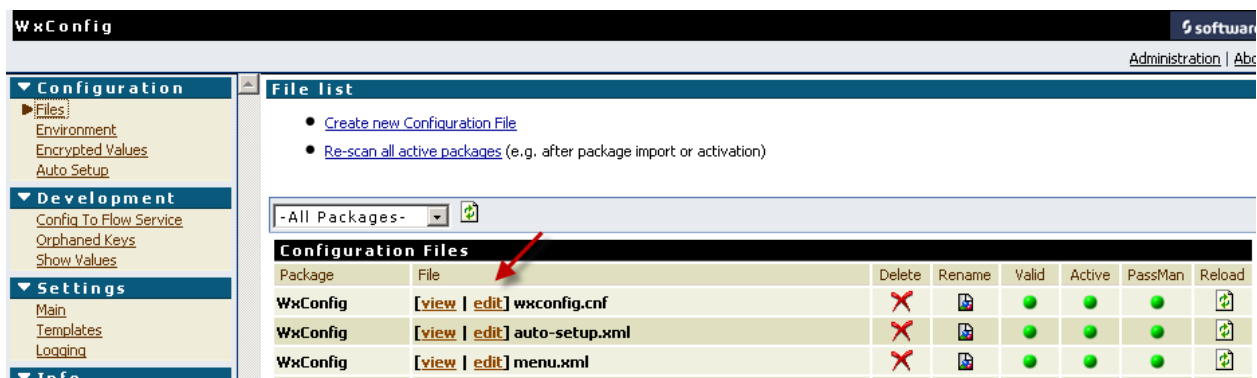
4.1.2 Section: Development

You can generate flow steps to an existing flow to read the configured values and get access to them within your flow service.
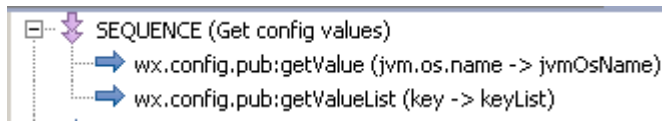
Orphaned Keys are listed to keep your packages clean and running

4.2 XML Configuration

An often underestimated capability of WxConfig is the use of XML configuration. While property files are great for many use-cases, XML can be used to model the configuration of entire entities (to avoid the term "object"). And since XPath expression can used to retrieve data, one can easily search for specific entries. In addition it is possible to retrieve not only discrete values for elements or attributes but entire structures as Integration Server documents (IData).

The following example will demonstrate the use of XML configuration. Let's assume we are dealing with a financial application that at some stage needs to aggregate figures from various contries into regions. So we need a mapping that tells us to which region a country belongs. Traditionally this kind of information was often persisted in a relational database system (RDBMS). In its simplest form there would a table (MAP_COUNTRY_REGION) with two columns (COUNTRY, REGION) and at some point a query was issued imilar to `select * from MAP_COUNTRY_REGION where REGION = 'EMEA';`.

One can also solve this with WxConfig using property files. An example would look like this

```
countries.EMEA=Austria
countries.EMEA=United Kingdom
countries.EMEA=Germany
countries.NA=United States of America
countries.NA=Canada
countries.NA=Mexico
...
```

It would be fairly easy to get all countries for EMEA by invoking the service `wx.config.pub:getValueList` with `countries.EMEA` provided as key. In reality, though, there will not only be countries but quite a few more values that need to be configured per region: responsible manager, controller, etc.; people allowed access to financial data (usually further segregated); required margin below which an alert will be triggered; and many more. While somehing like this could still be achieved with a properties-based configuration, it would be quite cumbersome in many aspects: To get a list of all defined regions, one would need to retrieve all defined keys for the package using `wx.config.pub:???` and then filter the results. Likewise, it would not be trivial to get the region of a given country.

It is much easier to model this in XML and then use XPath.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <regions>
        <region name="EMEA''>
            <manager>Mike Smith</manager>
            <countryList>
                <country>Austria</country>
                <country>United Kingdom</country>
                <country>Germany</country>
            </countryList>
            ...
        </region>

        <region name="NA>
            <manager>Daniel Meiers</manager>
            <countryList>
                <country>United States of America</country>
                <country>Canada</country>
                <country>Mexico</country>
            </countryList>
            ...
        </region>

    </regions>
</config>
```

To get all available regions the XPath expression `regions/region/@name` fed into `wx.config.pub:getValueList` is enough. And all data for a given region (e.g. EMEA) can quite easily be acquired using the service `wx.config.pub:getDocumentList` with `regions/region[@name='EMEA']`.

So XML configurations are a good match for many things that traditionally would have been stored in a database. Of course, this works only for static, non-transactional data. If occasional changes happen, this can still be implemented with WxConfig, if the lack of transactions (and ACID properties) is acceptable. Changes can be applied using the web UI, modifying files directly, or by using CRUD services.

4.3 Working with Environment-Specific Settings

4.3.1 Basic Environment-Specific Settings

A very common requirement is to have configuration values that are specific to an environment (DEV, TEST, PROD, etc.) and/or host. (It is usually preferable to not bind the applicability to a host but a logical environment.) The respective parameters are described in section 7.2 and also the config file (`wxconfig.cnf`) that comes with the WxConfigpackage.

The approach is the following:

- Think about which of your configuration values are environment-specific and which not.
- The ones that apply to all environments go into the package's `wxconfig.cnf` file
- The others go into `DEV.cnf` (this name is just a suggestion, use whatever makes sense to you).
- Do the same for your other environments and adjust the values in the files respectively.
- Configure each of the environments to its value using either the GUI or editing environment.cnf in the Integration Server's main configuration directory.
- For XML there is one additional step to do. Given that the package's initial/default configuration file (`wxconfig.cnf`) must be properties-based, you also need to create a separate file (e.g. `wxconfig.xml`) and source it in (`wx.config.incl=wxconfig.xml`). For the rest, just use `DEV.xml` instead of `DEV.cnf` as file name.
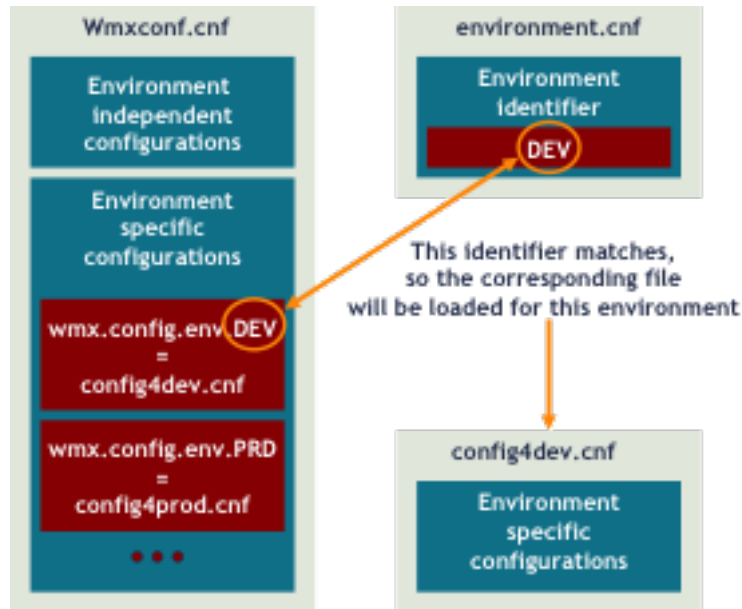
Figure 4.1: Diagram to illustrate the environment specific behavior

The following instructions show how to do this in practice:

- Assuming that we need three environments we will need three environment specific configuration files. Use WxConfig to create a file for development, named as `dev.cnf`, one for the user acceptance test called `uat.cnf` and one for production named as `prod.cnf`
- Start by creating `dev.cnf`

- Add whatever values you need, we use `dbName` as an example

- Save the `dev.cnf` file, repeat the steps above but create the `uat.cnf` file using the template function:

- When you now edit the `uat.cnf`, you will see that it is a copy of the template / `dev.cnf` file:

- Now edit the `uat.cnf` and replace `dbDevelopment` to `dbUAT`. Do the same steps as explained above to create the `prod.cnf` file with the key `dbName` configured as `dbProduction`.

- You now should have three config files for three possible environments that should look like these:

4 Fundamentals
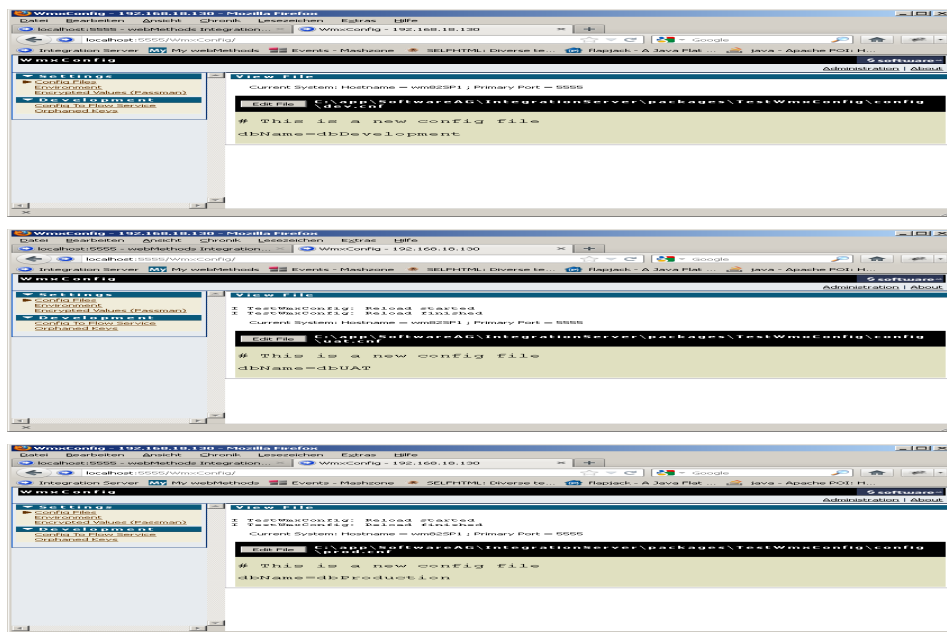
- The package determines the type of the system from the content of the file $SAG_HOME/IntegrationServer/
  config/environment.cnf . This file, if it does not exist, is automatically created through a startup
  service of the package

- To change the value one can either use a text editor and go through the file system or use the
  built-in web GUI (which is the same that is being used to change the content of configuration files).
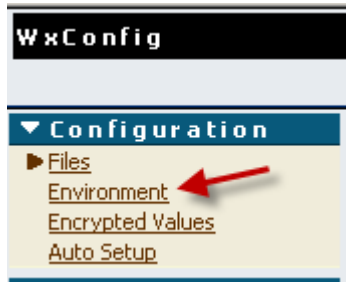
- Switch to edit mode and change the initial value from CHANGE to DEV
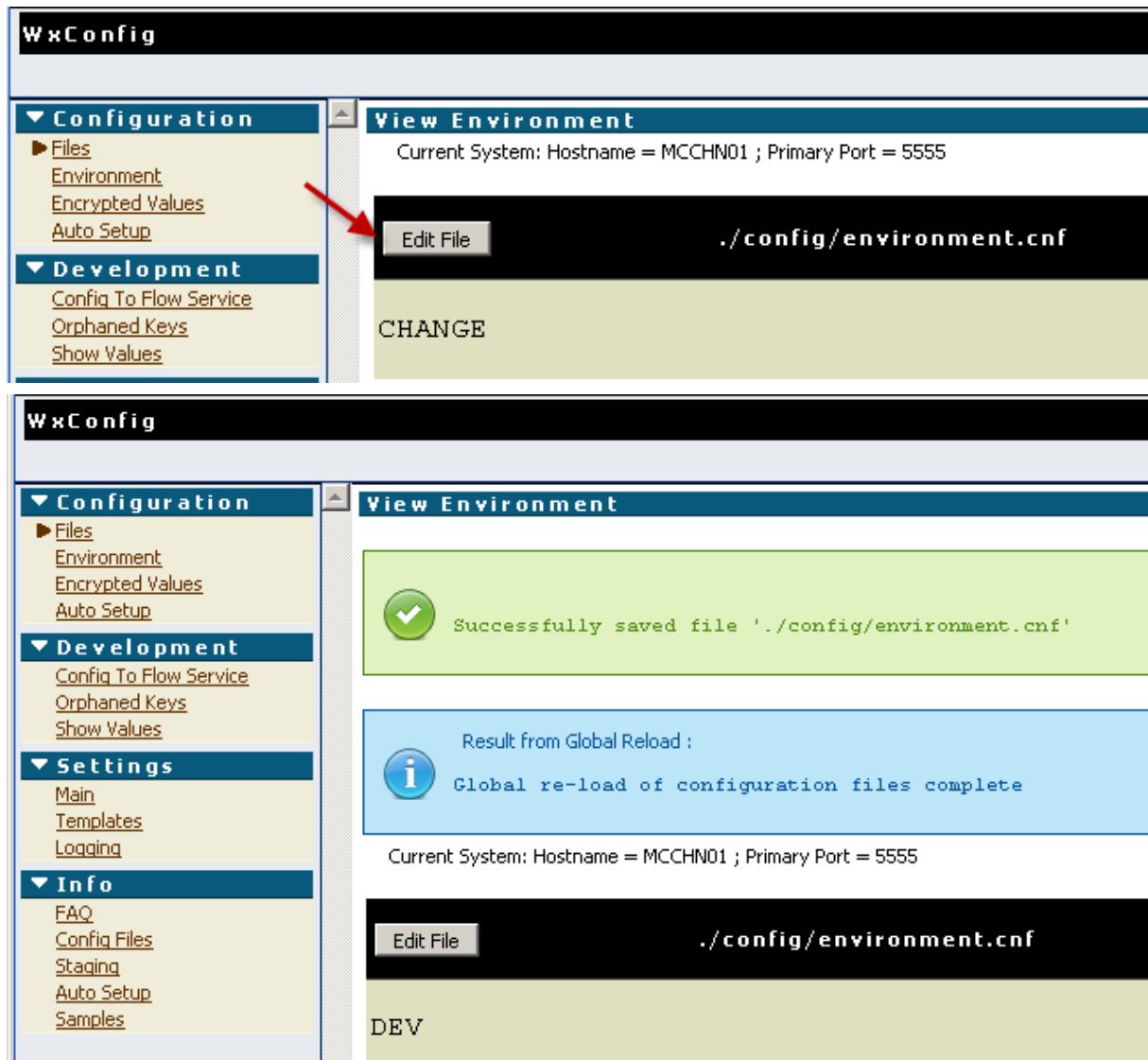
The entry "DEV" works like an index which tells **WxConfig** to what environment this system belongs.

To demonstrate the behavior, do the following:

- In section 3.2.3 (Getting rid of orphaned keys) you had changed the property `key1` to `key1a`. Make
  sure to revert this back to `key1`

- Enrich the flow service to read our new, environment specific variable `dbName`:

Refresh the service within your designer and run the service. The result should look like that:
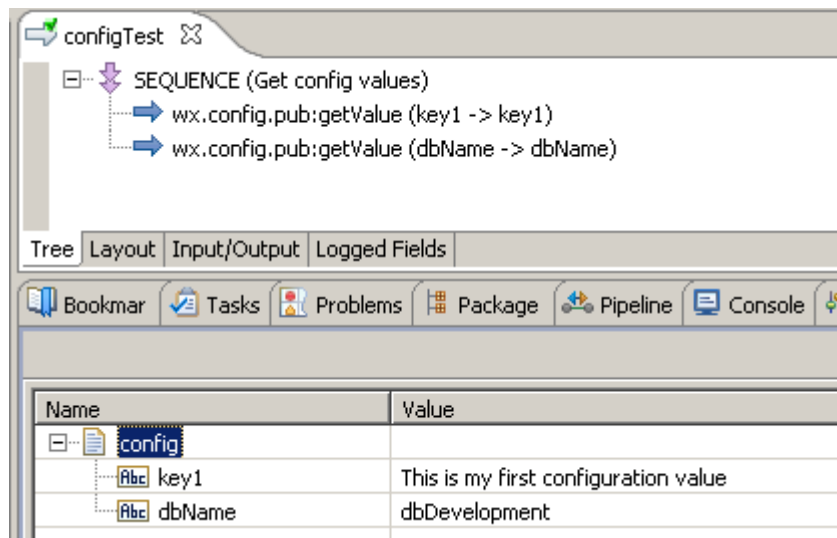
If you get an error like below, you probably forgot to change `key1a` back to `key1` in `wxconfig.cnf`

- A Now change the environment.cnf to point to UAT and save:
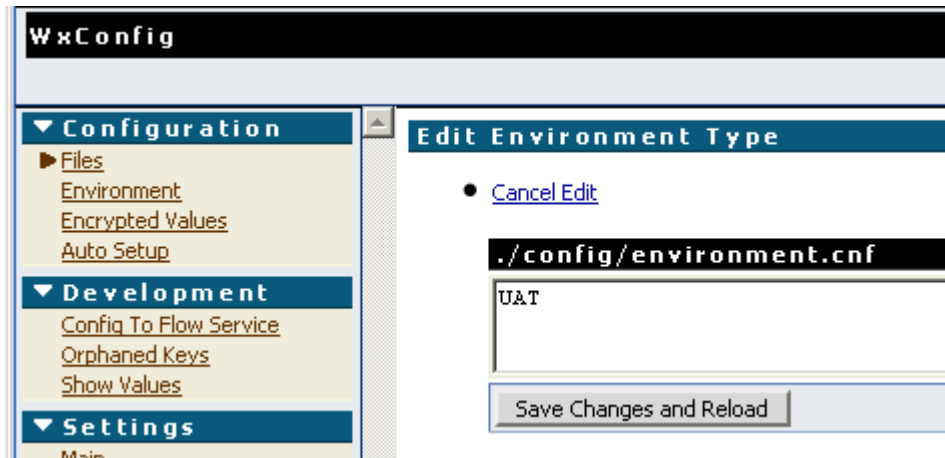
- You have to perform a rescan to refresh the values after a change of the environment type(which should not change that often in real environments):
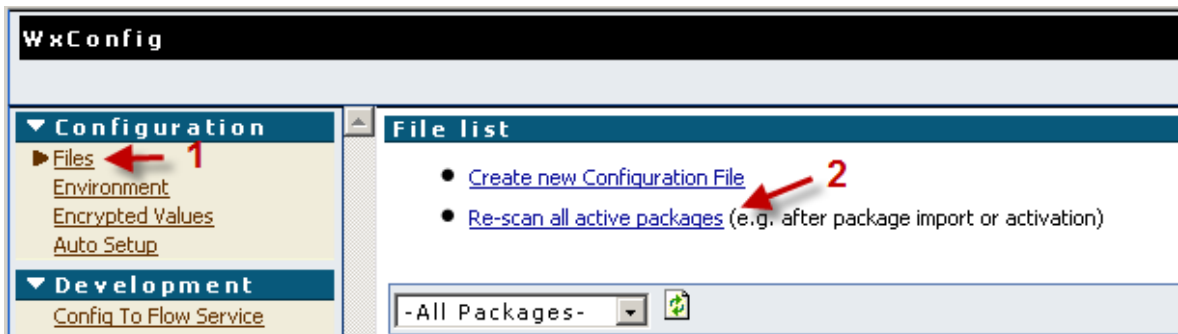
- Run your service within designer once again and check, that the configuration will now point the `dbName` to `dbUAT`:
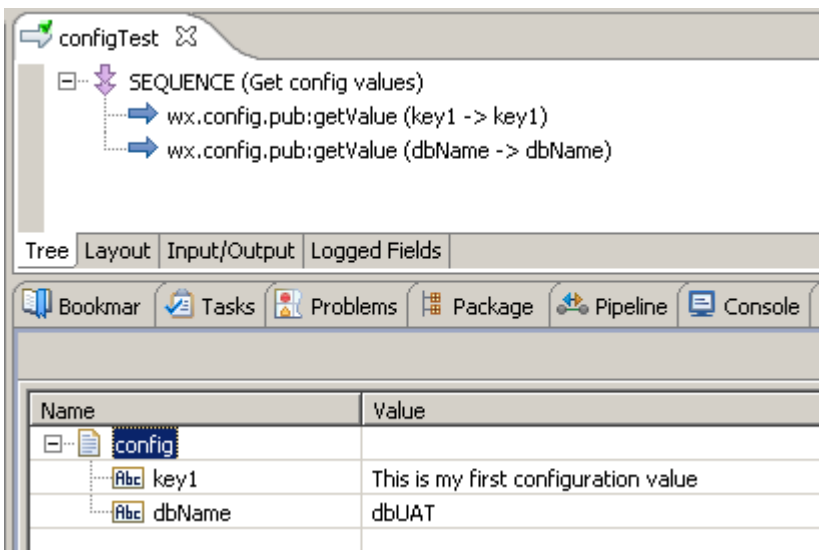
- Change the environment.cnf to be PRD, refresh the configuration, run the service and verify that `dbName` now points to `dbProduction`.

4.3.2 Advanced Environment-Specific Configurations regarding Host and Port

The above presented procedure is one possibility to load configurations in a conditional behavior. With WxConfigyou have two more possibilities of dependent inclusion of configuration values:

- Host: wx.config.host.HOST = <<configuration>>.cnf
  WxConfig loads the referenced configuration file only if the system's name matches the parameter

HOST. With this option, in addition to the basic environment–specific settings, you can e.g configure different machines within a production environment.

- Host + Primary Port: wx.config.host.HOST_PRIMARY-PORT = <<configuration>>.cnf WxConfigloads the referenced configuration file only if the systems primary port matches the parameter HOST_PRIMARY-PORT. With this option you can handle different configurations on a machine with multiple instances e.g. for clustering.

Both options are available through the file creation UI. Also, they can be applied manually using a special syntax for the file inclusion.[Warning: Draw object ignored][Warning: Draw object ignored] To support your development, WxConfigdisplays the syntax at the bottom in the configuration file edit mode. It also displays the possible settings for HOST and HOST-PRIMARY-PORT (for copy paste usage):

Current System: Hostname = wm82SP1 ; Primary Port = 5555

Special Properties

wmx.config.incl=include.cnf

wmx.config.env.ENV=environment.cnf

wmx.config.host.HOST=environment.cnf

wmx.config.hostPort.HOST_PRIMARY-PORT=environment.cnf

You can also mix all three options to handle dependent configurations. But keep in mind, that the last valid configuration will be delivered by WxConfig.

To clarify what this means, remember that our three files `dev.cnf`, `uat.cnf`, and `prod.cnf` contain the same keys with different values and imagine, that the environment is set to DEV and that the `wxconfig.cnf` looks like the following:
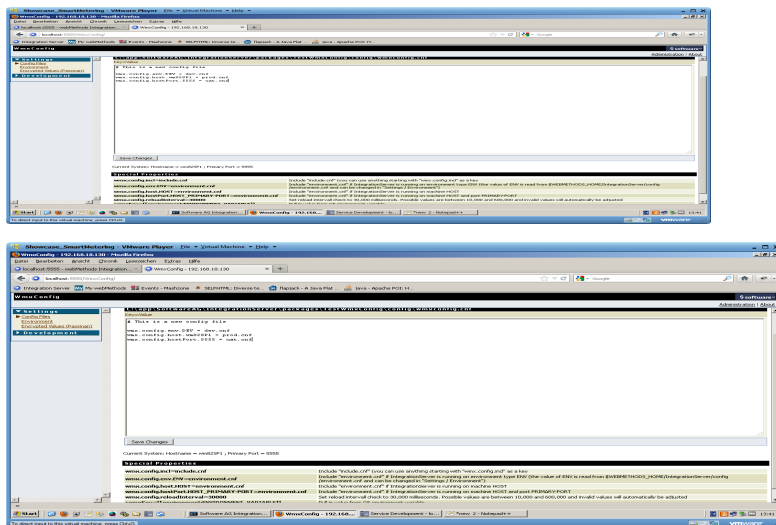
These assumptions will force WxConfig to deliver the values of `uat.cnf`, because it is the last entry and the systems primary port is 5555. The configurations for the host and for the environment are also true, but they are top to the primary port clause.

If you would read the keys as a value list, you would receive all values.

4.3.3 Management

Compare values across environments

4.4 Conditional Loading

Similar to environment–specific values it is possible to load files based on other conditions:

- Type of operating system (OS): Windows, UNIX/Linux, or Linux: Typical use–cases are different paths, JVM settings that are somehow dependent on the OS (e.g. number of threads working well), etc.

- Version of Integration Server: This can e.g. handle differences between extended settings (or their defaults), adapters, etc.

- Date/time: When values should automatically change at a certain date/time, this can be done. Examples would be new VAT rates at the start of a new calendar year, a new set of discounts for the holiday season in retail, etc.

5 Encrypted Values

It is a regular requirement to handle sensitive information (usually passwords) within configuration files and WxConfig supports this using the built–in password manager (also known as PassMan) of Integration Server. What basically happens is that the configuration file does not contain the value for the key anymore, but holds a reference (that has to follow a naming convention) that will be resolved against PassMan.

Syntax: key=[[encrypted:KEY_FOR_ENCRYPTED_VALUE]]

The value of KEY_FOR_ENCRYPTED_VALUE must NOT contain a semicolon, which is reserved for internal handling of secure cross-package interpolation. Explicit use of a semicolon will lead to WxConfig not being able to resolve values.

On the main screen there two places where you can see references to encrypted values.
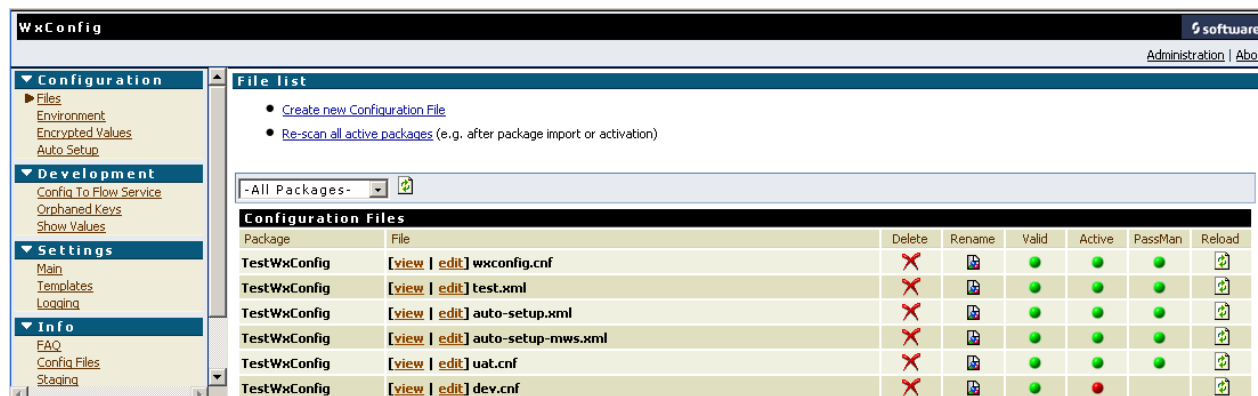
The column *PassMan* in the list of configuration indicates whether you need to do something. When the underlying logic retrieves the list of configuration files, it also scans them for the use of encrypted values. And if one such value is not properly defined on the current system, that is shown as a red sign.

The other place, where it is also possible to define, update, and remove encrypted values is the menu *Configuration / Encrypted Values*, which will be explained in more detail in the next section.

5.1 Setting or Updating an Encrypted Value

Introducing a new encrypted value is a two–stage process:

- Add an entry to a configuration file that points to PassMan
- Set the actual value in PassMan

As starting point here is a detailed set of instructions:

To keep things simple, let's add a new value to your configuration file `wxconfig.cnf` .Go to the list of configuration files and click on the one you want to extend.

- Enrich your WxConfigfile to store the keys for `dbUSR` and `dbPWD` like shown in the figure below:
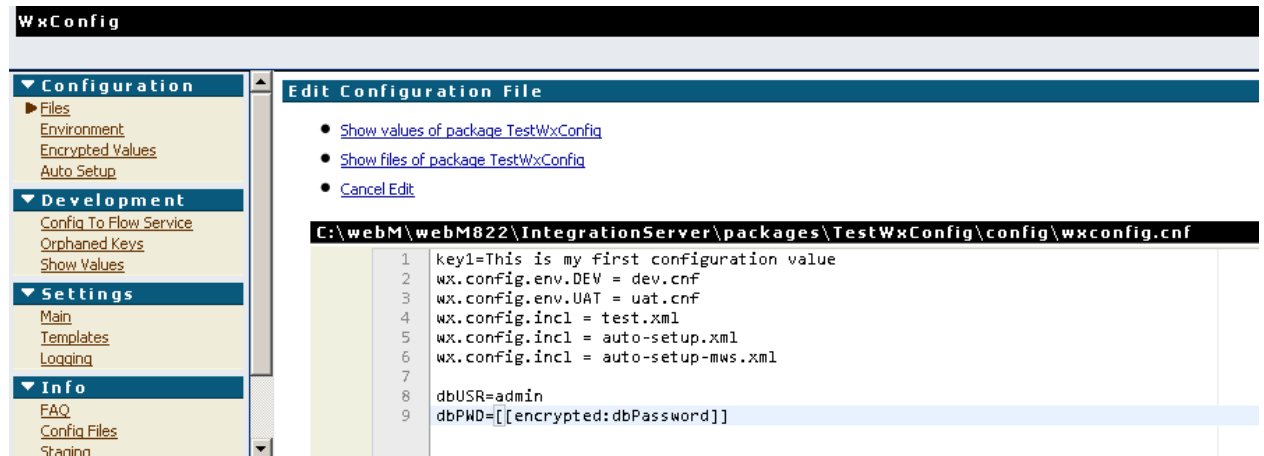
- After saving your changes go back to the file list. The list indicates with a red dot, that you have not set the encrypted value in the PassMan store:
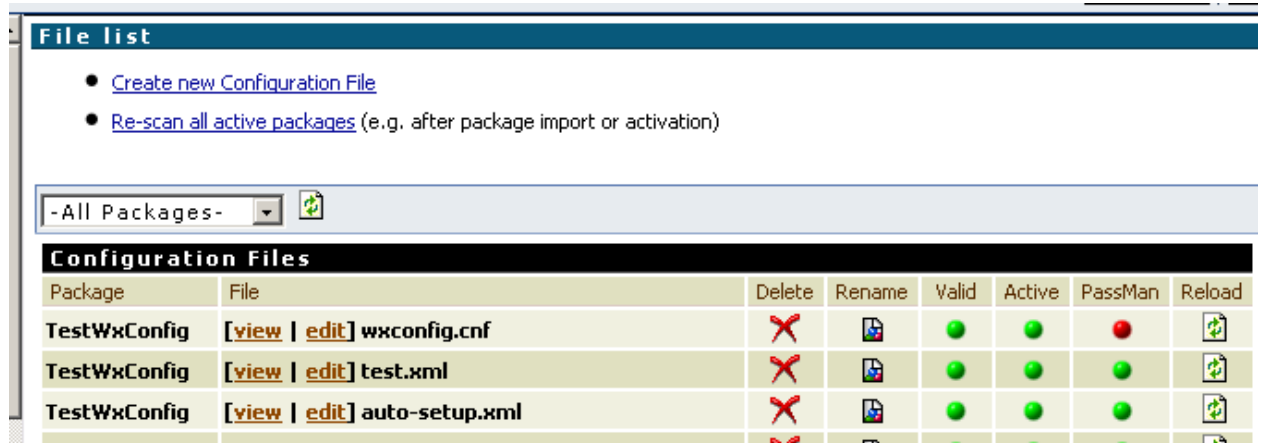
- For a more detailed view, just click on the *Encrypted Values (PassMan)* link on the left-hand side. The following overview comes up:

The red icons in the Status column indicate that the password does not exist yet in the local PassMan.

- To create it, and also for later changes, just click on the respective entry in the "Encrypted Values" column. Enter the value (e.g. "manage") twice (to avoid typos) and click *Apply Changes*.
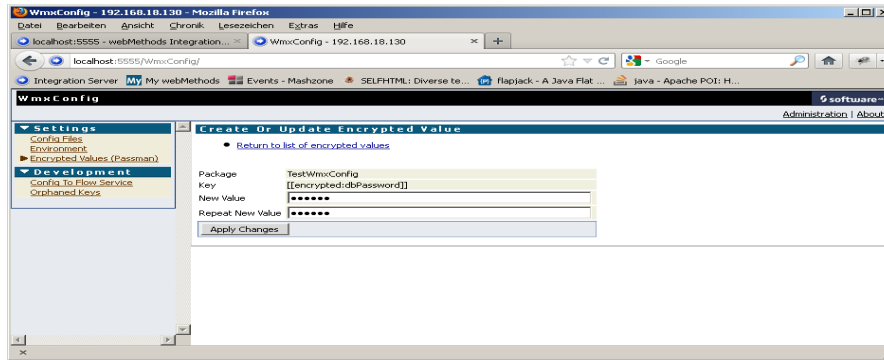
- The following screen will show the output, and there will also be a corresponding entry in the server log file. Please also note that the status of the first password has changed from red to green.
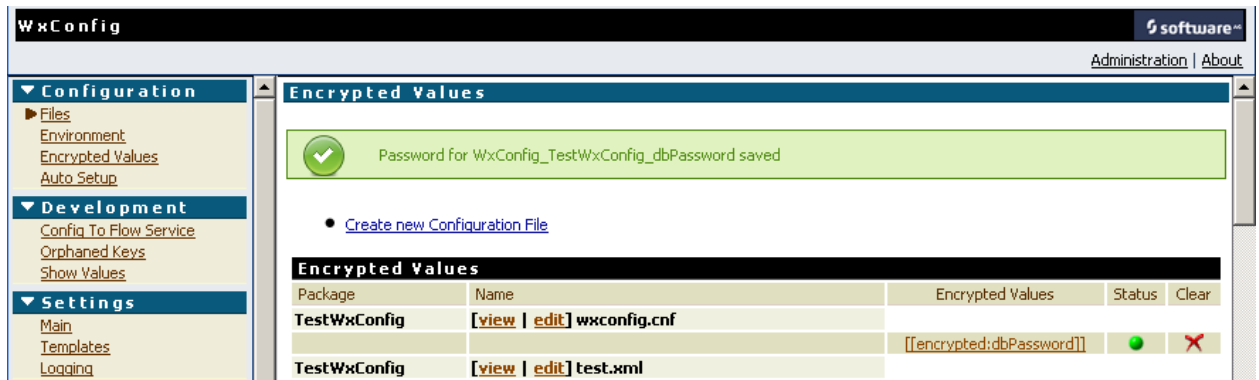
If you look at the message closely, it will also make clear how the naming convention in the configuration files maps to what gets created in PassMan:

WxConfig:PACKAGE-OF-CONFIGFILE:KEY-IN-CONFIGFILE

- Check the behavior by enrichment of your flow using the build in function *Config to Flow Service'*, refresh and run your service to see the result:

5.2 Deleting an Encrypted Value

- To delete a password from PassMan you just need to click the "Clear" icon next to it

and confirm this.

- Finally you should see the following result:
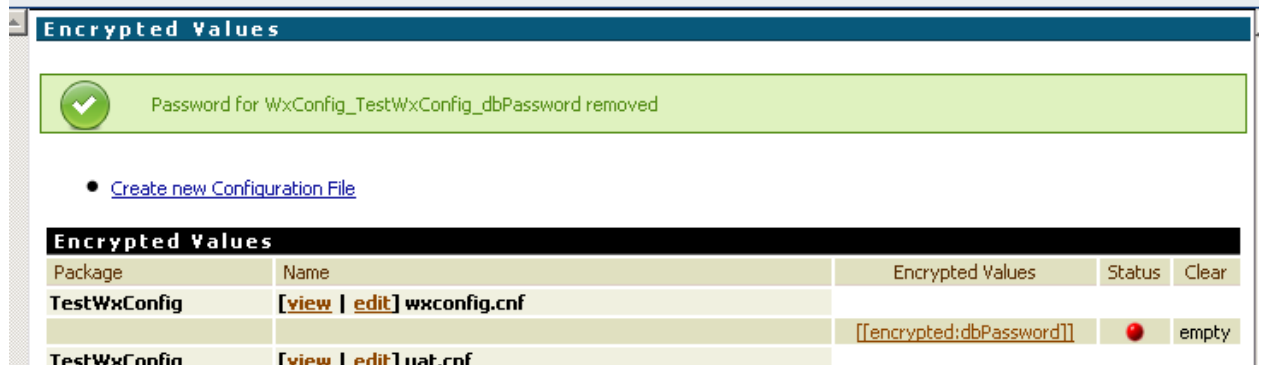
Please do now set the encrypted value again, otherwise the following steps in the guide might not work.

5.3 Unused Handles

If handles are found that are not referenced from the existing configuration files any more, they will be listed below the normal list, and can also be deleted from there. This avoids having sensitve information lying around in the system without anyone taking notice.

The same applies to values that are from either disabled packages or packages that do not exist any longer.

5.4 Descriptions

Since v1.7 it is possible to add a description to each encrypted value. This is especially useful when there is a handover from the developers who introduced encrypted values, to those that need to configure/set them in higher environments. The latter are typically operations people who often have a different view: The developers choose the names of encrypted values from an application perspective (e.g. `DB_HOST_CUSTOMER_USER1_PASSWD`), whereas the operations people think in infrastructure terms (e.g. Password for database user `USER1` on `cluster1.db.prod.int`). In older versions of WxConfig this was often solved by either providing separate documentation or using a naming convention that would also allow operations staff to easily grasp the meaning behind an encrypted value.

The description can be provided via UI (recommended) or by manually adding a property that follows the required naming convention (`encrypted.value.desc.ENCRYPTED_VALUE`). When done via the UI the description will be added to file that defines the encrypted value.

The description field comes with full support for variable interpolation. So when the password is stored for e.g. a database connection that is environment–specific, it is highly recommended to add something like "Password for database user ${db.user} on ${db.host}"

5.5 Automation

The aforementioned manual approach to applying actual values to the encrypted values should only be used in a development environment. All other environments (CI, TEST, etc.) should handle this in an automated fashion.

5.5.1 Bulk–Import from File

There is an export/import capability (links at the top of the screen) that forms the basis for this automation. The invocation can happen in the following ways:

- During WxConfig startup
- During package startup (i.e. usually triggered by package reload)
- Through a REST call

A typical workflow would be like this:

- Navigate to *Configuration / Encrypted Values* and click *Bulk–Import of Encrypted Values*

- Download file or copy–paste contents to editor for necessary changes. The actual values from the current environment are not contained in the file for security reasons.

  Instead there are placeholders ("???"). For a successful import, it is required to replace these placeholders with the actual passwords.

- For automatic import during startup of WxConfig this file needs to copied (name must be `WxConfigEncVal_*.txt`) to the path specified by the location alias `BULK_IMPORT_ENCRYPTED_VALUES`.

  By default it points to `./config/WxConfig/EncryptedValues`, which resolves to `$SAG_HOME/IntegrationServer/instances/default/./config/WxConfig/EncryptedValues`

  To change the file location go to *Settings / File Location Aliases* in the web UI of WxConfig and click the edit icon next to the current value.

- At startup of the WxConfig package all files matching the search pattern `WxConfigEncVal_*.txt` will be picked up, processed, and renamed afterwards to prevent duplicate processing.

  The new name contains the date/time when it was processed and `SUCCESS` or `FAILURE` to indicate the result of the operation.

  (Further details can be determined from the WxConfig log file (`$SAG_HOME/IntegrationServer/instances/default/logs/WxConfig.log`.)

It also possible to have the bulk–import triggered when a package's configuration gets loaded by WxConfig. Two parameters exist to control this:

- `wx.config.encryptedValues.bulkImport.runOnPackageLoad={true|false}`

  If set to `true` (default: `false`), when a package's values get loaded into WxConfig, there will also be a bulk-import of encrypted values from files (see next paragraph for selection of files).

- `wx.config.encryptedValues.bulkImport.fileFilterWildcard=EncVals_MYPKG*.txt`

  This wildcard expression controls what files are being processed during package loading bulk–import of encrypted values. This avoids picking up files that belong to other packages and should be processed at a different time. If not specified, the default (`WxConfigEncVal_*.txt`) will be used. During startup of WxConfig itself (as opposed to loading the config values for a package), this value has no effect; in that situation the aforementioed default kicks in.

In addition to have the import started during package reload, it is also possible to trigger it via a REST call. The actual logic that gets executed is exactly the same, only the invocation path is different. The main reason for this additional way is that a package reload disables the package, and all the others that depend upon it. This can be avoided by using the REST call.

URL: `http://<hostname>:<port>/rest/wx/config/encrypted/import/files/<PACKAGE_NAME>`

5.5.2 REST Services

Encrypted values can be set via a REST service. This is different from the previous section in that the former allowed to trigger the import of a file. Whereas what is described here facilitates the direct change of a value, without any file being involved.

The primary motivation to introduce this direct change capability, in addition to the bulk-import files, was its use in the context of general–purpose configuration management systems like Chef, Puppet, Ansible, etc. While it is possible to set password through a file as an intermediary, it complicates the whole process. One needs to write a template with a placeholder for the actual value, create the actual file with the correct value in it, and then trigger the import.

After a while it became obvious that a more direct approach would be beneficial. The obvious choice was to provide a REST service interface.

5.5.3 Locking

Changes to encrypted values can be prohibited by creating a lock file.

6 Variable Interpolation

6.1 Within Package

It is possible to reference already defined properties when declaring new ones. This prevents duplication of information, which often leads to problems.

Syntax: `newKey=${EXISTING_KEY} additional contents`

Example: `tmpFileWithPath=${tmpDir}/file1.tmp`

The reference variable can be located in any file containing configuration values. It also possible to reference a variable that was defined in a property file from an XML file and vice versa.

6.2 Across Packages

While the interpolation is normally limited to one package, it is possible to use it across packages with a special syntax

Syntax: `newKey=${pkg:PACKAGE_NAME;EXISTING_KEY} additional contents`

Example: `showCfg=Show WxConfig is ${pkg:WxConfig;showWxConfigFiles}`

While the aforementioned approach can be used within a configuration file, the same can also be done in a Flow servic. There one can access variables from other packages by explicitely requesting them via the respective services (`getValue`, etc.).

6.3 Environment Variables

Similar to the encrypted values, you can leverage existing environment variables from the operating system.

To do so, the syntax used is:

`someKey = ${env:ENVIRONMENT_VARIABLE}`

- You have to replace the "ENVIRONMENT VARIABLE" with one of the systems known keys. An example for MS Windows is given below:
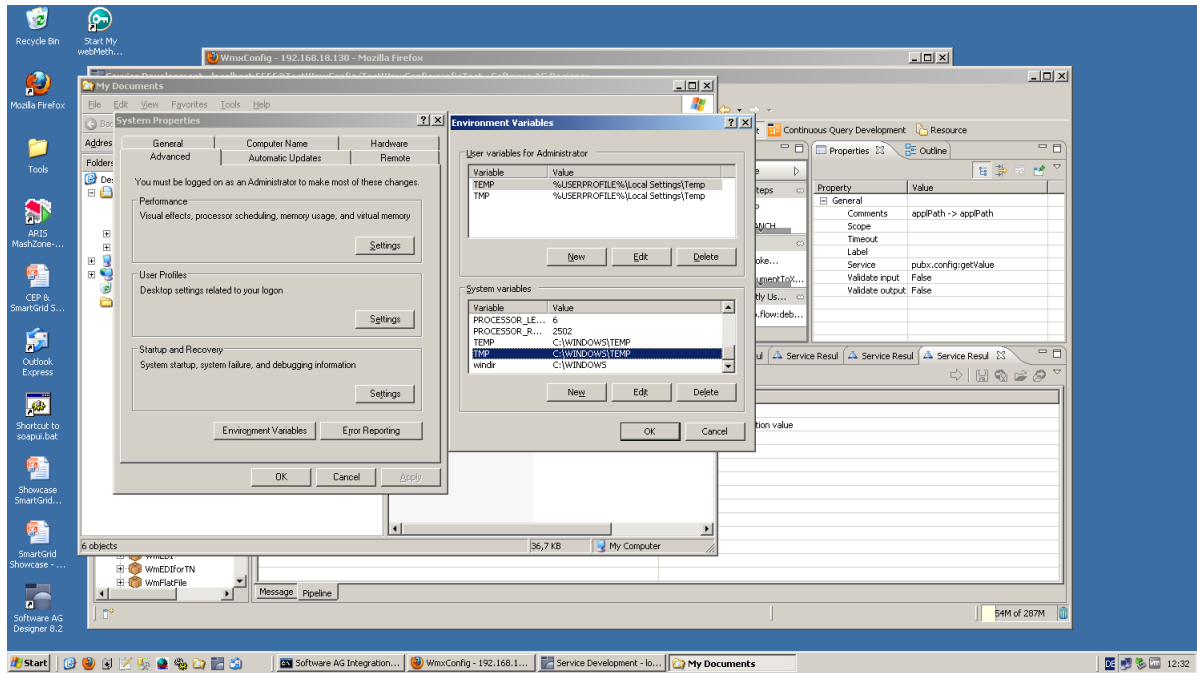
- Add the new variable using the *Config to Flow Service* functionality
- In Designer you will receive the tmpPath at runtime:
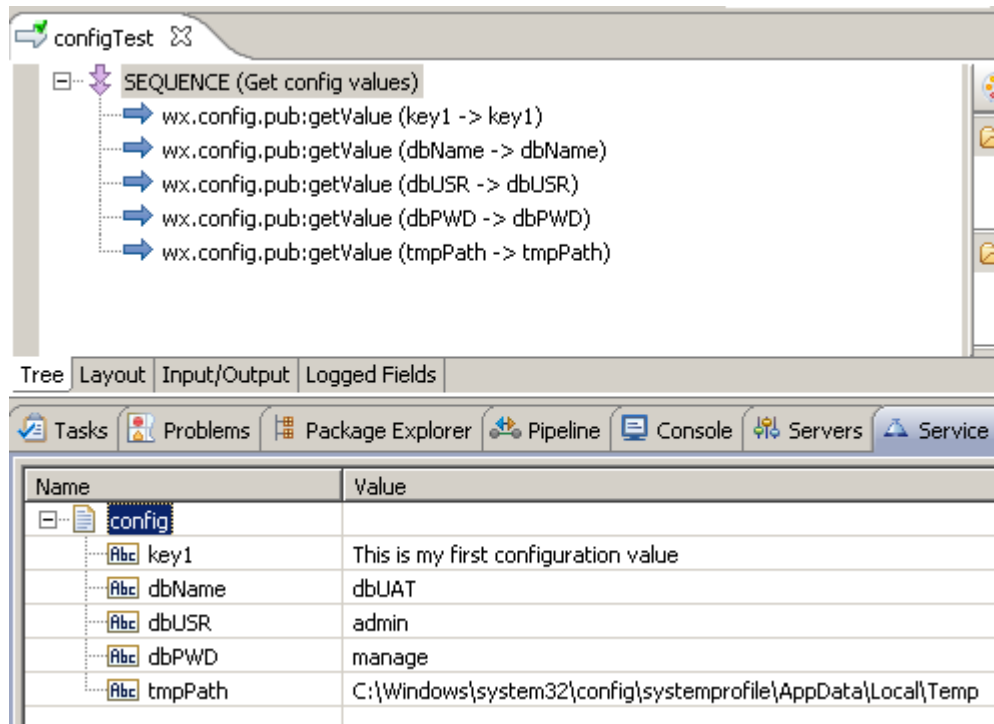
6.4 File Contents

One can pull the contents of an entire file into a configuration variable.

Syntax: `newKey=${file:FILE_NAME_WITH_PATH}`

Example: `propertyTemplate=${file:c:\temp\template.txt}`

The file's contents are not cached and each access to the variable will trigger a file read access. So this is not a suitable approach for a high–load scenario.

6.5 Service Result

One can pull the result of a service execution into a configuration variable.

Syntax: `newKey=${service:SERVICE_NAMESPACE;inputName1=inputValue1;`
           `inputName2=inputValue2;...}`

Example: `concatString=${ wx.config.pub.samples.scenario_02_variableInterpolation:`
           `d_serviceInvocation;input1=AAA;input2=BBB}`

The example will produce "AAA_BBB"

The service needs to provide its output in string field called *result*, so that it can be picked up.

6.6 JVM System Properties

One can pull JVM system properties into a configuration variable.

Syntax: `newKey=${sys:PROPERTY_NAME}`

Example: `operatingSystem=${sys:os.name}`

6.7 Current Date/Time

One can pull the current date/time into a configuration variable.

Syntax: `newKey=${date:}`   for default format

       `newKey=${date:FORMAT}` for custom format

Example: `currentDateTime=${date:yyyy-MM-dd_HH-mm-ss.SSS}`

The format string complies with java.text:SimpleDateFormat; please see http://docs.oracle.com/javase/6/docs/api/java/tex for details.

6.8 Public Constants from Java Classes

One can pull public constants from Java classes into a configuration variable.

Syntax: `newKey=${const:PACKAGE.CLASS.CONSTANT}`

Example: `propertyTemplate=${const:com.softwareag.wx.is.config.`
       `directive.FileInclusionDirectiveConditionDateTime.DATE_TIME_FORMAT}`

7 Miscellaneous

7.1 Samples

Samples are provided in the WxConfigSamples package that can be installed from the WxConfig UI.
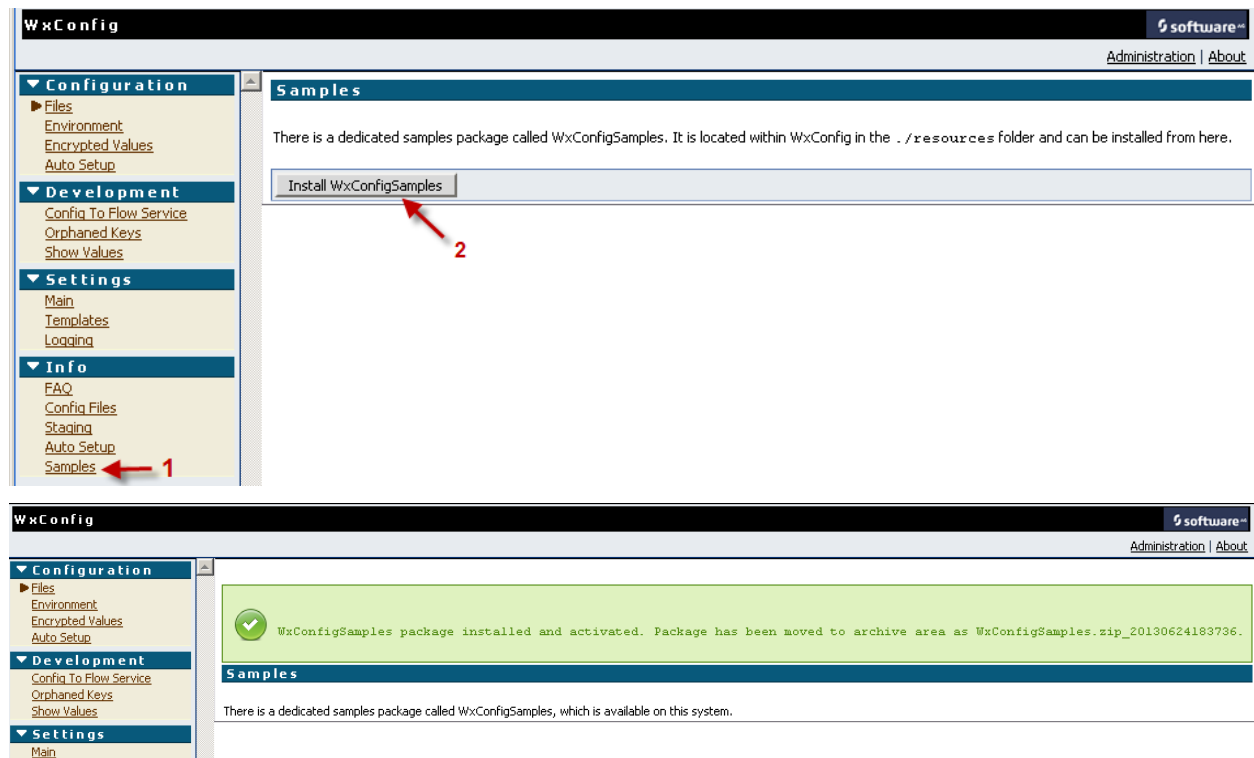
Please step through (debugging mode) all examples and compare the results with the input parameters and the contents of the configuration files

There is one central configuration file for **WxConfig** that handles general things like the period after which configuration files are checked for changes and re-loaded if needed (technically this value can be overwritten on a per-package basis, although it is probably not needed very often). All other values should be handled by files that are placed within the packages that use them. This will keep inter-package dependencies at a minimum and allow for a clean deployment approach.

All configuration files support a mechanism to include other files, either in general or environment-specific. So it is easily possible to keep different "aspects" of configuration values in their respective files. These included files can be located at an arbitrary position in the file system, thus allowing for great flexibility. However, having them outside the package will require extra care when deploying.

7.2 Special Parameters

All parameters are also documented in the UI, either the *Info / Config Files* menu or below every file when editing it.

wx.config.incl=include.cnf Include `include.cnf` on all systems (unconditional loading)

wx.config.incl.inactive=clearTables.sql Show `clearTables.sql` in file list, so that it can easily be edited, but without trying to add its contents to the configuration in memory

wx.config.excl=ports.cnf Exclude file `ports.cnf` from automated inclusion as inactive file

wx.config.env.*ENV*=env.cnf Include file `env.cnf` if Integration Server is running on environment type *ENV* (the value of *ENV* is read from `$WEBMETHODS_HOME/IntegrationServer/config/environment.cnf` and can be changed in *Settings / Environment*)

wx.config.host.*HOST*=host.cnf Include file `host.cnf` if Integration Server is running on machine *HOST*

wx.config.hostPort.*HOST_PRIMARY-PORT*=host5566.cnf Include file `host5566.cnf` if Integration Server is running on machine *HOST* and the primary port is configured to *PORT*

wx.config.reloadInterval=30000 Set reload intervall check to 30000 milliseconds. Possible values are between 500 (use with care!) and 600,000 and invalid values will automatically be adjusted. Applicable only for the "ON_CHANGE" strategy

wx.config.reloadStrategy={ON_CHANGE|NONE|MANAGED} Set reload strategy to check on file changes ("ON_CHANGE", default), never reload ("NONE"), or explicitly trigger reload ("MANAGED"); the values are not case-sensitive and will be converted to upper–case internally.

wx.config.autoSetup.execute={true|false} Turn automatic execution of auto-setup on or off for package. If set to "true", the auto-setup will be executed PRIOR to loading the package by a custom PackageListener in a fixed order (in technical terms, the service `wx.config.pub.autoSetup:_all` will be executed). If a different order is needed, the execution must be done by a custom startup/shutdown service, which calls whatever required services in the `wx.config.pub.autoSetup` folder.

wx.config.encrypted.skipLogForMissingHandle=updatePassword Do not log a missing encrypted value for the handle defined by `[[encrypted:updatePassword]]`. Each handle needs to be specified separately. The list is specific to the package in which it is placed.

wx.config.interpolator.checkUnresolved.excludeFile=fileName Skip one or several (if property is provided multiple times) files when checking for unresolved interpolators via the service `wx.config.impl.admin:checkUnresolvedInterpolators`. The list is specific to the package in which it is placed. The exclusion does not affect the display of files in the WxConfig web UI.

wx.config.incl={ALIAS=TEMP}include.cnf Include `include.cnf` that is sitting in the location defined by the file location alias *TEMP*. See menu *Settings / File Location Aliases* for more details. This approach is generally recommended over the use of paths, because it is a cleaner mechanism to abstract system specifics away from the package content.

wx.config.env.DEV={PATH=c:/tmp}include.cnf Include `include.cnf` from the directory `c:/tmp`. This option should only be used in combination with a conditional include (environment type or host name). Otherwise there is a high risk that the included file is not available on all systems. In general, the use of location aliases should be strongly considered instead of using paths. Paths (full and relative) are supported for config files when using the `{PATH=...}` directive; details depend on underlying OS.

wx.config.incl=[dateTime=2012-11-27_07:50:00;2012-11-27_07:51:00]include.cnf Include `include.cnf` for the defined timeframe. If current date/time is before the start, a scheduler entry will be created. Also, a scheduler entry will be created for the date/time of deactivation. THIS FEATURE IS EXPERIMENTAL !

wx.config.incl=[OS={windows|linux|unix}]include.cnf Include `include.cnf` if the operating system matches the host running IS. The OS name is not case-sensitive. THIS FEATURE IS EXPERIMENTAL !

wx.config.incl=[wmVersion=8.2]include.cnf Include `include.cnf` if the version matches that of the running IS. It is evaluated from least to most specifc; i.e. you can specify "8" or "8.2" or "8.2.2". THIS FEATURE IS EXPERIMENTAL !

wx.config.incl=[globalValues=true]include.cnf Include `include.cnf` as global values.

THIS FEATURE IS EXPERIMENTAL !

someKey=[[encrypted:KEY_FOR_ENCRYPTED_VALUE]] Maintain actual value inIntegration Server's encrypted password store (aka PassMan). The value of KEY_FOR_ENCRYPTED_VALUE must NOT contain a semicolon, which is reserved for internal handling of secure cross-package interpolation. Explicit use of a semicolon will lead to WxConfig not being able to resolve values. See section 5 for details.

wx.config.dev.keyCheck.ignorePackage=PACKAGE_NAME Exclude an entire package permanently from the orphaned keys listing (only useful in the main wxconfig.cnf of the WxConfig Package)

wx.config.newFile.ignorePkg=PACKAGE_NAME When creating a new config file, this package will be excluded from the drop-down list for the target packages; can provided multiple times to define several packages with one package per line

someKey=Value is ${otherKey} Use value from other key in same package

wx.config.dependency.create.skip={true|false} Instruct WxConfig that it should *not* create and/or update a dependency to itself altogether. A typical use-case would be that WxConfig serves as an editor to files (e.g. SQL scripts), but does not provide any functionality other than that. So on systems that have WxConfig installed, there is an added value, while the package with e.g. the SQL scripts does not get "tainted" by an automatically created dependency to WxConfig; the latter would make it unusable on systems that do not have WxConfig.

wx.config.dependency.update.skip.anyVersion={true|false} Instruct WxConfig that it should *not* change an existing, not version-specific depedency. So if package "A" already has a dependency to *WxConfig (*.*)* the latter will not be touched. The net result will be that package "A" runs with all versions of WxConfig. Until v1.3 WxConfig created these non-version-specific dependencies. From v1.4 on, the dependency will match the current version of WxConfig. So WxConfig v1.5.x will create a dependency to *WxConfig (1.5)*, and also, by default, update existing dependencies (e.g. *WxConfig (1.4)* after an update from v1.4) to this value.

8 Auto–Setup

Quite often a package relies on the existence of configuration settings that are maintained outside the package for the entire Integration Server (e.g. JMS connection alias). When such a package is then moved to another server instance, normally all those settings need to be re-applied manually. Apart from people often not documenting those things at all, it is cumbersome and error-prone. The approach WxConfig takes is to have those things in a special configuration file within the package. Using a special package listener that WxConfig' brings with it, or a custom startup service (only recommended when a special order is required for executing the various setup activitis) the custom package can then ensure that whatever is needed, gets set up automatically. The approach is defensive here, i.e. if something already exists it will not be changed, unless explicitly configured.

The following configurations can be created

- Schedulers (one–time, simple and complex)
- JMS incl. connection alias, client groups, topics and queues (Universal Messaging and Broker)
- Ehcache cache mangers and caches
- Extended settings
- Adapter connection for adapters based on WmART
- IS users, groups, ACLs (incl. applying them to services and entire folders)
- MWS user, groups, roles (requires WxMWS package)
- Allowed paths for file operations from pub.file:* services
- Web Service endpoint aliases
- Proxies
- JDBC pool and functional aliases

The following steps outline the configuration of a simple scheduler. They are equally applicable for any other configuration item.

- Create new config file for auto-setup
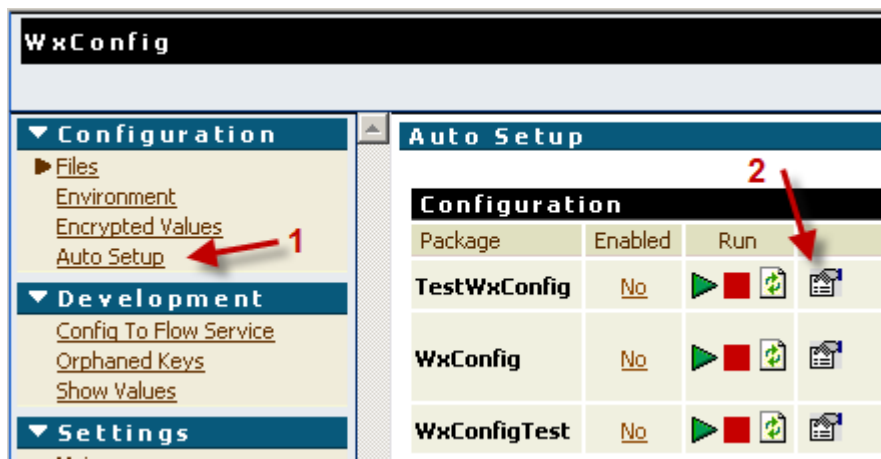
   In the file–creation dialogue a number of templates is available, each covering a particular area. For this exercise, choose `template-as-scheduler.xml` and leave the rest of the values as they are pre-populated.

Start editing the file

- Edit this file in such a way, that you delete every node but the scheduler node. In addition place valid entries like shown in the figure below and save
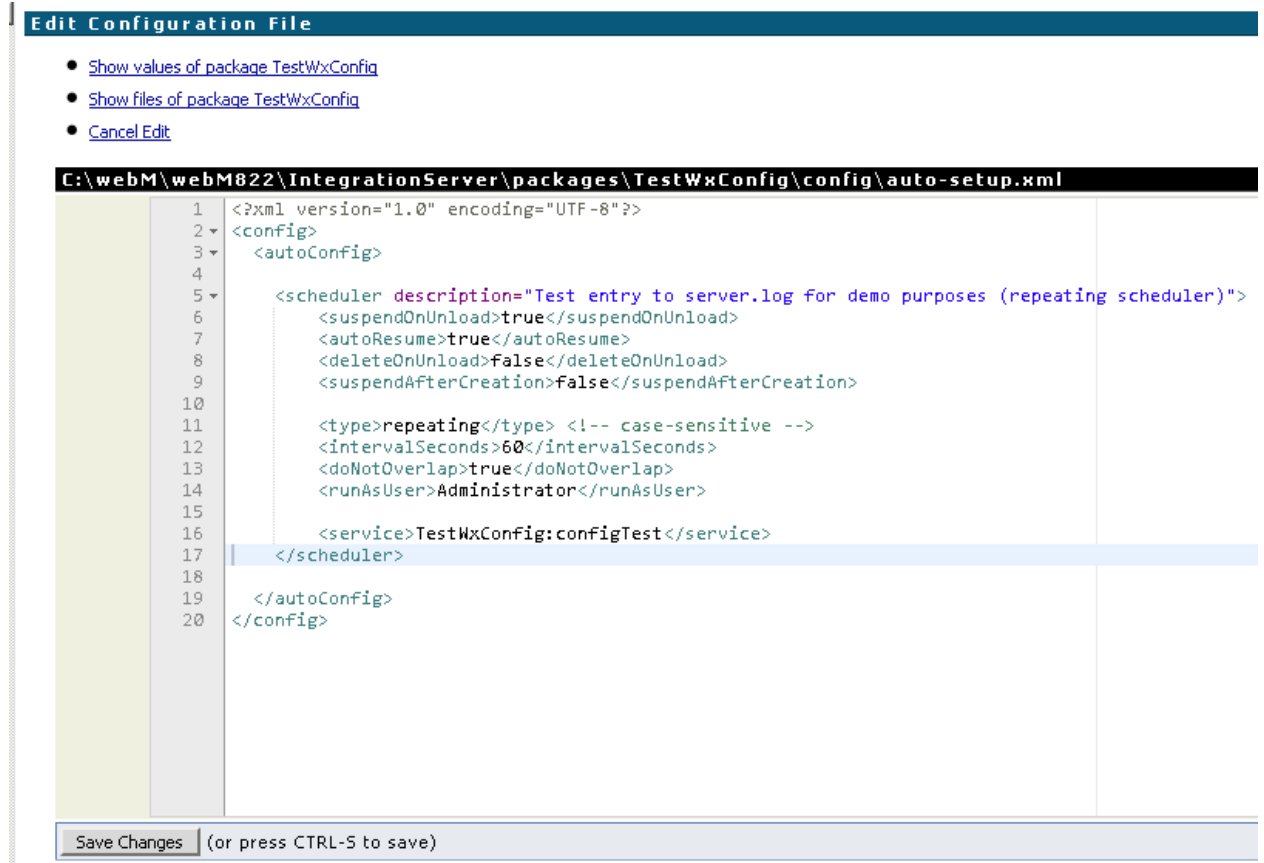
8 Auto–Setup

- Activate auto-setup for the TestWxConfigpackage
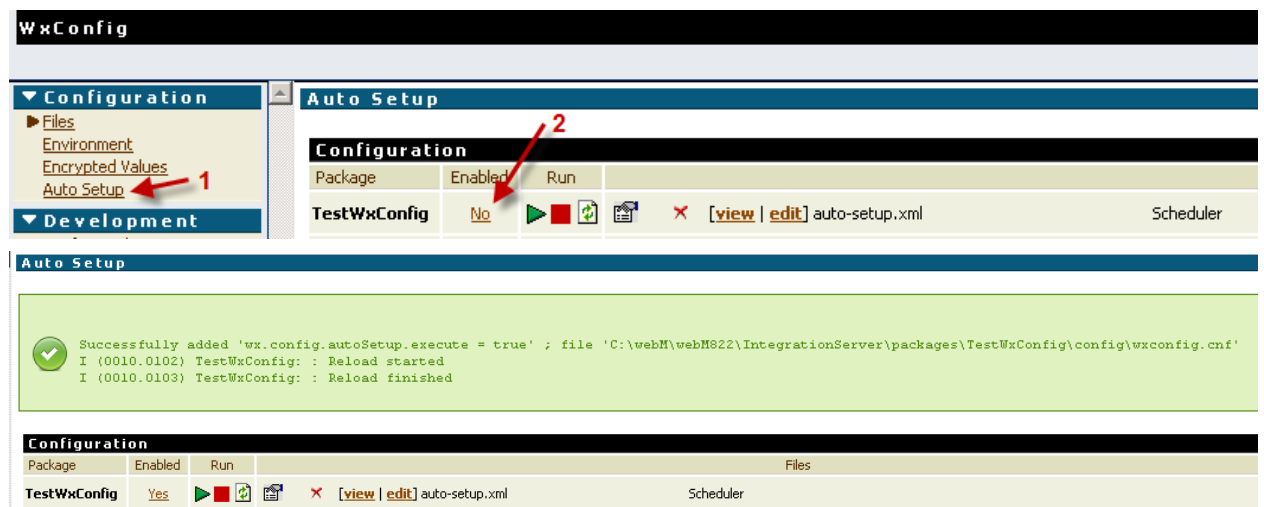
- Test the new auto-setup configuration by simulating a package startup

- Check log file `WxConfig.log`
- Check that the scheduler was created properly

The scripting of the other settings work in a similar manner and are not further explained here.

Every setup scripting has a built-in mechanism, that ensures an existence check. If the object to be created already exists, it will not be touched. This means, that changes to the objects might be done by deleting them (e.g. the scheduler), adopt the configuration within the auto-setup.xml followed by an execution of the creation service. This ensures that manually changed configurations are not changed automatically.

The advantage of scripting these configurations comes when you want to deliver packages and need to ensure that certain configurations exist.

9 Advanced Topics

9.1 Security

WxConfig defines a number of ACLs and groups with corresponding names. These ACLs control what parts of WxConfig a user has access to. There is a general–purpose ACL called `WxConfigStandard` that is used for all parts, which do not need special treatment.

9.2 Startup

In addition to the somewhat simplified description of WxConfig's startup behavior in section ???, you find in–depth coverage below.

Hidden file

Automatic loading of *.cnf, *.xml, *.properties as inactive

9.3 Reloading

It is possible to deal with changes to files that happen outside of WxConfig(also called out-of-band changes). This is usually relevant in one of the following scenarios:

- Development: The developer changes files' contents to test various routes in the code.
- Testing: As part of test automation files' contents are changed by scripts.
- Regular deployment: It is common to changed configuration files' contents as part of a deployment procedure. Many production systems run in a clustered setup and store their configuration files on a shared location (typically SAN or NAS). This way, all cluster instances get their configuration from the exact same source.
- Correct value in production: If a wrong configuration value somehow made its way onto the production system, many organizations prefer to first change it there, consciously skipping the normal deployment procedure, and only later correct it in the downstream systems.
- Disaster recovery: As part of disaster recovery activities it is common to re-configure the connections applications have to other systems. Those updated configurations will then point to backup instances of the other applications. All this preferably happens in a fully auotmated fashion to avoid manual errors and limit the knowledge required to execute the operation as much as possible.

To be able to deal with all those scenarios successfully, a number of features are available in WxConfigthat provide the flexibility to deal with the different requirements.

- Individual files can be reloaded explitely in the UI or via a service invocation.
- By default, configuration files are monitored for changes in a configurable interval (default 60 seconds, adjustable between 500 milliseconds and ???) and transparently re-loaded in case a change is detected. An entry will be made to the log file.
- Within each configuration file it is possible to change that behavior and either turn reloading off entirely, or require an explicit action to do it.
- In the UI one can switch an entire package into update–mode, which means that file monitoring will temporarily be disabled completely to avoid inconsistent readings. Once all changes to files are completed, the update–mode is finished and things are turned back to normal.

- The update–mode operation mentioned above can of course also be dealt with using services to support a fully automated approach.

9.4 Global Values

A very common requirement is sharing configuration values across packages. Typical scenarios are:

- Truly global values that represent organization–wide information (e.g. company name, hostname of central mail server, etc.)
- Values are used across packages
- Define a global default that can be overridden by packages if necessary.

9.5 Java API

WxConfig also contains a Java API. So one does not need to invoke services from Java but can access configuration directly. There are two starting points for this.

- `Access.java`: Dedicated accessor class with various static methods, which should be the preferred approach
- `WxConfig.java`: Main class of WxConfig which is where all calls from `Access.java` fundamentally end up. While all reasonable efforts are being made to keep this stable, it cannot guaranteed that all changes will be backwards–compatible.[1]

9.5.1 Basic Usage

The simplest and usually most often used functionality is to retrieve a single string for a given package.

```
String value = Access.getProperty(key, packageName);
```

Both parameters are strings and key can also be an XPath expression for XML configuration.

Likewise, a string array is fetched with

```
String[] valueArray = Access.getProperties(key, packageName);
```

Both methods, which only perform the most basic operation, are wrappers around the fully fleteched version:

```
String[] value Array = Access.getPropertyValues(key, configID,
                           noServiceException, pipeline, ignoreGlobalValues,
                           scanAllConfigurations, noVariableInterpolation,
                           showEncryptedValues)
```

Further operations are (for more details check the JavaDoc documentation):

updateProperty Update value of a given property; also works for discrete values in XML

remoteProperty Remove given property; also works for discrete values in XML

addProperty Add property with fine control to which file it will be added; also works for discrete values in XML

getConfigFileForKey Get file in which a given property is defined

getDocumentList Return entire parts of XML configuration

addDocument Add Integration Server document (i.e. `IData`)

removeDocument Remove part from the XML configuration; can be used to implement update functionality in conjunction with addDocument

---

[1] Up to v1.5 this class was named `WxConfigTree.java`, which had become an increasingly misleading name and was finally changed in v1.6.

getConfigFileList Get list of configuration files

getFileObjectForName Retrieve file object by name or path

9.5.2 Advanced Usage

For full access the `Access` class is not sufficient. Instead one has to use `WxConfig`, for which `Access` is basically a wrapper. `WxConfig` is also internally the main class of the package. Given that the implementation uses the singleton pattern, everything needs to go through `WxConfig.getInstance()`.

For a complete understanding it is necessary to read the JavaDocs, but a basic overview can be found below.

WxConfig

The following methods are a good starting point for using WxConfig's Java API. They all require to retrieve the singleton instance via `getInstance()` upfront.

- `getConfig(String)`: Retrieve the configuration for a given Integration Server package as an instance of `PkgConfig`. The object returned contains more than just the values. In particular it allows access to the underlying files with their content, conditions for loading, etc. This is by far the most important method in `WxConfig`.

- `getAllPackages()`: Return all packages using WxConfig

- `forceGlobalReload()`: Triggers a complete reload of all files. Technically this removes all relevant data from memory and then executes the same internal method that is run when the package starts. This method should *never* be used on a system with in–flight transactions as there is no locking for read–access.

PkgConfig

Represents the configuration for an Integration Server package. In addition to a combined view onto the values from all loaded configuration files, it allows access to all files (represented by `ConfigFileList`) and to the controll for special deployment options.

This class is derived from `CompositeCfg`, which in turns is derived from Apache Commons Configuration's `CompositeConfiguration`. Therefore the various methods for accessing the configuration values are not repeated here. It is perfectly fine to use them for read access. Write access is *strongly* discouraged, though, since it will only modify values in memory (but not on disk). Instead the respective methods from `Access` should be used.

The main methods are:

- `getConfigFiles()`: List with all configuration files of the package. This also includes files which are not loaded, because not all conditions (e.g. environment type, operating system, etc.) were met.

- `reload()`: Reload all files for this package, but not the entire system.

9.5.3 Miscellaneous

Naming Conventions

The package has evolved over more than seven years of active development. Therefore no *strict* naming convention exists. This is especially true when the name of on Integration Server package needs to be provided. One can find `cfgId` (current standard), but also `cfgID`, `id`, etc. This will gradually be unified in future versions.

9.6 Package Dependency

9.7 Clustering

Deployer–support

Shared file system via location alias

9.8 Auditing

A common requirement is, especially for production system, to record any changes applied. This is commonly referred to as auditing and WxConfig can handle it in two ways:

- Subversion plugin for the UI
- Built–in auditing

9.8.1 Built-In Auditing

At its core the built-in auditing (as opposed to using Subversion or other version control systems) works such that for every change it takes snapshots of the changed files, one before the change and one after it. In addition, a file with metadata in XML format is created that describes the change(s) in a way that can be evaluated by a program. After a configurable number of days those files are moved to an archive location, where they are aggregated by year and month. And once the retention period has expired, files are deleted from the archive location.

Configuration

Auditing is configured via an XML configuration, for which a UI is available.

The following things can be observed from the screenshot above:

- Auditing can be configured globally with exceptions for inividual packages. So either auditing is generally active but turned off for some packages; or it is generally inactive but turned on for some packages. Prior to v1.7 changes to the default configuration require a reload of WxConfig to become effective.
- The default configuration resides in the server instance's configuration directory (`$SAG_HOME/IntegrationServer/instances/default/config/WxConfig`, which also ensures that an update of WxConfig does not change customized values. Package–specific configurations are stored in the packages' `./config` directories.
- The approach of a global default and per–package exceptions is also followed for archiving and retention duration.
- Default values:
  - Audit files: `$SAG_HOME/IntegrationServer/instances/default/logs/WxConfig/audit`

- Number of days before archiving: 2
- Archived files: `$SAG_HOME/IntegrationServer/instances/default/logs/WxConfig/audit/_archive`
- Number of days before deletion: 4000

Audit Data

Each audited event is effectively a change to a single configuration file. It can be caused by one of the following actions:

- Interactive editing of file in the web UI
- Programmtic change via one of the following services
  - `wx.config.pub:addValue`
  - `wx.config.pub:removeValue`
  - `wx.config.pub:updateValue`
  - `wx.config.pub:addDocument`
  - `wx.config.pub:removeDocument`

For every audited event three files will be created: file before the change, file after the change, description of the change. The name format is the following

`<ORIGINAL_NAME_INCL_SUFFIX>_dateTimeStamp.<TYPE>`

where `<TYPE>` can be `old`, `new`, or `info`.

The audit files will be organized per Integration Server package, so underneath the location for audit files there will be one sub–directory for every package that has audited events. Those sub–directories will be created on the fly, if needed. So it is perfectly ok if some packages, although auditing is enabled for them, are missing. It just means that no auditing has happened yet.

Archiving and Deletion

To deal with a potentially large number of audit files, an archiving mechanism exists. It means that after a configurable number of days the audit files will be moved to a separate location on the file system. In this archiving location they will also be grouped by year and month. So in there will actually be two levels of directories: the package name and the year/month.

Files exceeding a second age threshold will automatically be deleted from the archiving location.

Both the archiving and deletion will be run by a scheduler in Integration Server every 24 hours. All operations will be logged to `WxConfig.log`.

9.9 Publish to Ehcache

WxConfigcan automatically publish the configuration value that it holds in memory to an instance of Ehcache. This makes it possible that any application with access to the respective Ehcache can be configured via WxConfig. So WxConfig can be used as a central configuration management engine for any application. Particularly interesting will be use–cases where applications run on many different systems with a mix of system–/host–specific and more general settings. A good example will be IoT (Internet of Things) devices.

9.10 Logging

The package logs quite a few things to its own log file (`$SAG_HOME/IntegrationServer/instances/default/logs/WxConfig.log`). The thresholds can be configured in the UI.

When the license is about to expire, this will also be logged to `server.log`.

It is recommended to monitor the log file using a suitable system. To facilitate the configuration, WxConfig provides a catalog of all its log messages in XML, which can be accessed from the web UI at *Logging / Message Catalog*.

9.11 Automatic Updates

10 Built-in Services

The built-in services can be grouped in public and thus supported services and internal ones. Public services (located in `wx.config.pub:*`) will be kept backwards–compatible between minor releases of WxConfig. So a version is supposed to work the same way on all 1.x versions of WxConfig (it may or may not do so in a future 2.x version).

Internal services (located in `wx.config.impl:*`) on the other hand may change without notice at any time or even be removed, replaced by others, etc. The vast majority of services in WxConfig is internal and therefore not documented here. Nevertheless users are encouraged to scan through them as a means to better understand the possibilities of WxConfig. In a way the internal services are the reference use–case for many aspects.

10.1 Retrieving Values

10.1.1 wx.config.pub:getValue and wx.config.pub:getValue List

Return either the first value (`wx.config.pub:getValue`) or all values (`wx.config.pub:getValueList`) found for a given key or, in case of XML configuration, XPath expression. If you are sure that only one result exists or you want to get just the first one off a list, please use `wx.config.pub:getValue`.

key The key or XPath expression (in case of XML configuration) for which to return value

wxConfigPkgName Package from which retrieve data; usually detected automatically by inspecting the call-stack; can be override to get data from an another package (beware hidden dependencies!)

noServiceException If nothing is found, return `NULL` instead of throwing a `ServiceException`

ignoreGlobalVariables When no match is found in the package, as a fall-back WxConfig looks into the global values (virtual config package GLOBAL)

substituteVariables If the returned value contains references to the pipeline (e.g. `%orderDoc/poNumber%`) those get resolved against the pipeline at run-time; very useful for templates, esp. in conjunction with the `${file:xxx}` interpolator

configFile Only use data defined in a particular file, rather than the entire data of the package

noVariableInterpolation Setting this to true will disable variable interpolation

showEncryptedValue By default encrypted values are returned; setting this to false will return the handle instead (`[[encrypted:handle]]`)

scanAllConfigurations Only relevant if elements of lists are spread over multiple files; by default (this is inherited from Apache Commons Configuration) all parts of the in–memory configuration (so–called `CompositeConfiguration`) will be iterated over and once a match is found for the key, only entries in that part will be returned; technically this means that only entries from the first file (loading order as per WxConfig log file) that holds matching data will be returned; settings this to true will force WxConfig to go over all files.

10.1.2 wx.config.pub:getAllValues

Returns all values configured, either globally or for a given package. Mainly to be used for debugging purposes.

10.1.3 wx.config.pub:getEnvironmentType

Returns the actual environment type.

10.1.4 wx.config:pub:getServicesWithConfig

Lists all services with additional information, that take usage of WxConfig generated flow steps.

10.1.5 wx.config.pub:checkEncryptedValues

10.1.6 wx.config.pub:getDocumentList

Return part of an XML configuration as an Integration Server document list (also known as `IData`).

key The key or XPath expression (in case of XML configuration) for which to return value

wxConfigPkgName Package from which retrieve data; usually detected automatically by inspecting the call-stack; can be override to get data from an another package (beware hidden dependencies!)

noServiceException If nothing is found, return `NULL` instead of throwing a `ServiceException`

ignoreGlobalVariables When no match is found in the package, as a fall-back WxConfig looks into the global values (virtual config package GLOBAL)

substituteVariables If the returned value contains references to the pipeline (e.g. `%orderDoc/poNumber%`) those get resolved against the pipeline at run-time; very useful for templates, esp. in conjunction with the `${file:xxx}` interpolator

configFile Only use data defined in a particular file, rather than the entire data of the package

noVariableInterpolation Setting this to true will disable variable interpolation

showEncryptedValue By default encrypted values are returned; setting this to false will return the handle instead (`[[encrypted:handle]]`)

arrays Comparable to the same parameter in `pub.xml:xmlNodeToDocument`; returns an XML element as a list, even if only one occurrence is found (otherwise the list will be created automatically); only checks for the name of the XML element without its context, so if `<name>` is defined under `<company>` as well as under `<contact>`, both will be affected.