Application Intelligence Framework: A Praxis Taxonomy for Internet Developers

Waleed Nema
Windows Technical Planning and Support
Saudi Aramco*
PO Box 10092
Dhahran 31311, Saudi Arabia
Email: Waleed.Nema@Aramco.Com

Abstract

This paper investigated whether there is a universal standard for defining what a superior or intelligent software application is. It defined Application Intelligence via a set of proposed pragmatic measures defining a framework that can hopefully be used as a base for assigning a score card grade for any software application. Rather than measuring software quality, the discussion centered on how applications are able to declaratively define aspects about themselves, their environment, or certain business requirements in a standard way that middleware processing engines can understand and act upon. Such engines were also discussed.

The author offers an approach for designing smart applications by building features that enable them to communicate and report about operational and business parameters. At the same time, the supporting infrastructure is designed to deliver the value of data collected and act upon instructions posted.

The conclusion underscores that the contribution of the paper's ideas and approach are applicable to any software application with the emphasis being on Internet applications.

Keywords
Software application score card, software application management, Application Intelligence Framework, Message Processing Engine, runtime reporting, Business Synonyms, logging, alerts, Internet, web.

*The author expresses his gratitude to Dr. Ibrahim Mishari –Vice President Information Technology Services, and management of Computer Operations Department Mr. Ali Masari, Mr. Abdulaziz Otaibi and Mr. Nahed Rabeh for their support.

1 Introduction

For past decades, computer applications were mostly limited to human interaction and their own platforms [1,2]. It was not until the Internet and standardization technologies such as XML [3] that application-to-application communication and open systems software became a reality. Despite this openness and the advancement of computing technologies, what is still missing from the repertoire of today's Internet application design and management is a standard to drive the assessment of software applications on the business and operational levels [4].

This paper, then, attempts to provide an Application Intelligence Framework to standardize and optimize the relevant issues and to provide a praxis taxonomy which seeks to explain how designers of Internet applications can effectively and efficiently turn data into intelligence needed for business decision making.

Application Intelligence criteria consider issues such as configuration, metadata, messaging, logging, baselining, automation, changes, dependencies, health, performance, alerting, and reporting. The lack of some of these declarations becomes a factor in determining a grade of how intelligent an application is.

What is more, this approach may offer to designers and researchers a focus on how applications can become solutions to real-world business issues by providing complete infrastructure, facilities, tools, and reports to realize business benefits.

This paper is organized as follows: Section 2 provides a background of an Application Intelligence Framework, giving the foundations of a standardization approach; Section 3 defines the objectives of the proposed Application Intelligent Framework and the required processing engines. The next two sections, 4 and 5, discuss Application Management, both internally from within the application (section 4) and externally by managing dependencies and changes (section 5). Internal management covers the proposed folder structure for the framework, Application Runtime Reporting, jobs and the messaging engine, metadata, analysis reports, and optimization. Section 6 offers details of the ten elements of the praxis taxonomy and explains the finesse required by application designers to facilitate management and enhance services to construct a direct link to business objectives; and, the appendix includes 12 tables of XML declarations covering all presented aspects.

2 Background of Application Intelligence Framework

The focus of application development had been confined to the boundaries of application functionality with little attention to the management of the application or the resources it needs to be able to deliver a high quality service. In the cases when some of these issues were addressed, every application followed its own way and the same problem was probably solved multiple times within the same organization let alone the whole development community. Also in my experience, another problem in today's world of software development is the irony of creating an application that produces data that isn't presented to the business in useful format, in a timely manner, or being analyzed or checked for quality (more on Application Irony later).

Top

This paper takes an application centric view considering the application as a business service the availability of which is as important as the ability to provide business-relevant alerts and business-relevant data such as personalization or other usage information. Some of the issues are addressed by proposing standards and others through a list of 10 praxis taxonomy items. The praxis elements help demystify some of aforementioned challenges with distributed application design such as internet and web-service based applications.

In their recent Dynamic Systems Initiative [6], Microsoft is trying to completely describe a distributed system using the System Definition Model and build operational requirements into software development. This is great and certainly expands the view beyond a pure systems Total Cost of Ownership (TCO) approach but still remains in the Application-Systems domain. What this paper addresses, though, is a Business-Application-Systems approach where both business and systems considerations are addressed by application design and the support infrastructure.

3 What Is Application Intelligence Framework?

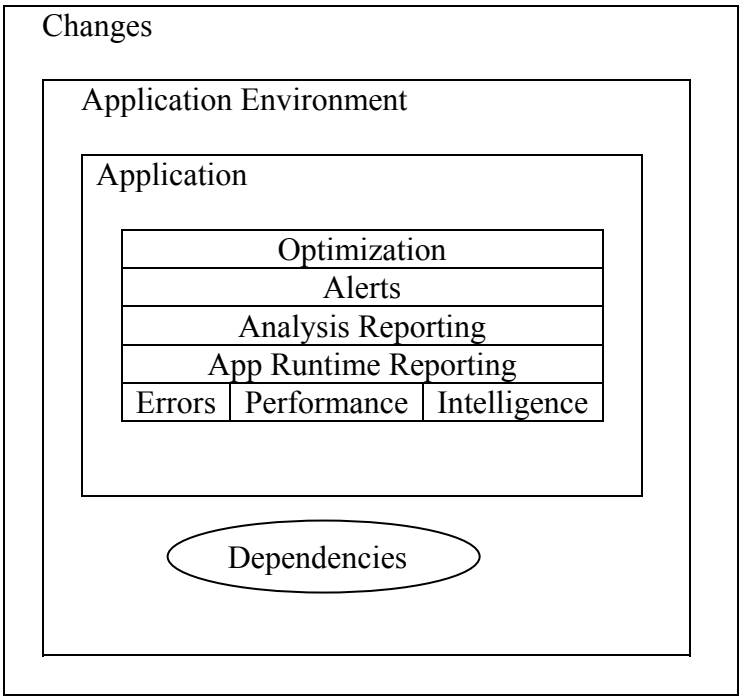


Figure 1: Application Intelligence Framework

Application Intelligence Framework (AIF) aims to standardize management and optimization of a software application in a way that yields maximum business benefit. It attempts to manage the application from within and manage its environment including dependencies and changes [4]

AIF is made of a set of specifications, automation tools, processing engines, and documentation guidelines. The Framework emphasizes building whatever reports or tools needed to directly make use of collected data that benefit the business need. Therefore, logging, baselining, reporting, and alerting are considered vital to AIF.

Top

3.1 AIF Objectives:

1. Enhance service availability by tracking changes and dependencies.

2. Build self-monitoring features in application design such as:
    a. Message/error reporting with actions and escalation.
    b. Proactive health monitoring.
    c. Self-performance tracking.
    d. Feature usage stats and personalization tracking.

3. Know and increase the level of business delivery through reporting, baselining, alerting, and optimization.

4. Ease management by creating Value Added Tools (VATs) such as administration front-ends, reports, Quality Assurance (QA) scripts, and automation. More than one layer of VAT may be needed in order to establish a direct link to business objectives.

5. Use a centralized folder structure (under the application root folder) as a configuration management database. Centralization of this information will help in configuration management, problem management, in addition to capacity planning and optimization

3.2 Engines:

The idea is to declaratively define the application environment in a generic way that would apply to any application and build specs for middleware processing engines to perform the specified functionality.

Required Engines:

1. Messaging Processing Engine (MPE)
   Works as automation and health monitoring engine as well. MPE is more fully described under section 4.2.

2. Logging including log reporting, baselining and alerting Engine (LE)
   LE is covered under section 4.2.1

3. Documentation Engine (DE)
   The functions of this engine are explained in section 6.9.

Top

4 Internal Application Management

4. 1 Folder Structure

The proposed folder/file structure fits under a folder called APP under the application's root folder. It is described in Appendix Table 1.

4.2 Application Runtime Reporting

Application Runtime Reporting (ARR) plays an important role in AIF where error messages, performance statistical messages, and usage intelligence messages could be logged. The Message Processing Engine (MPE) acts as a centralized message center that accepts messages from any system through a bin (ie. share, ftp mount point) thus acting as a corporate-wide message and automation center.

Messages support actions, scheduling, and escalation. Actions include: logging the message to a database, ASCII/XML file, or the event log; sending an email to a user or group; sending a popup window message to a user or a group; and/or running a job/command. Appendix Table 2 shows a sample message format. Appendix Figure 3 shows a schematic for the MPE.

4.2.1 Logging

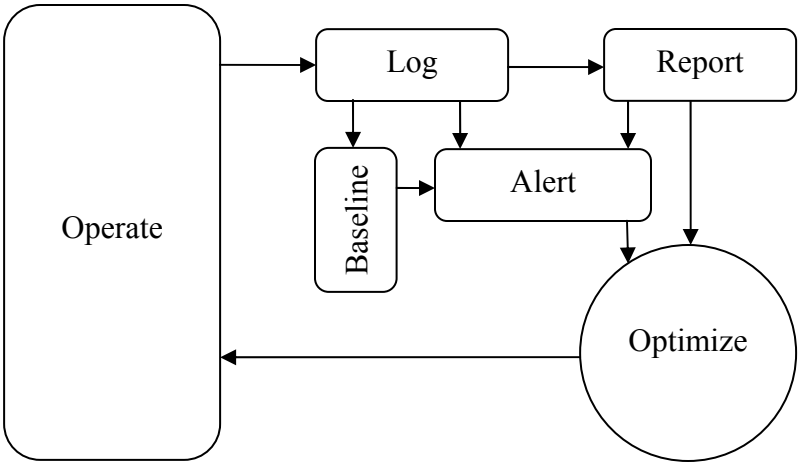


Figure 2: Complete Log Cycle

This diagram emphasizes the following:

1. The purpose of logging is optimization. Stopping anywhere before the loop is closed back to impact the current operation environment is pointless.
2. Log data becomes valuable with reporting and alerting based on pre-defined thresholds or based on an existing baseline.
3. Alerts can be based on log data directly, on reports, or on baseline data.

Logging is a form of ARR which could include error messages, performance data, auditing, or whatever event the application designer feels is important. Logging entries are normal messages that can support all the features mentioned above including actions, and scheduling. Log Config in Appendix Table 12 allows the application user to choose the fields and format of the log.

Top

Log Baseline Config:
This declaration file in Appendix Table 11 tells the Logging engine what types of baseline measures should be calculated.

Log Alerts Config:
This declaration file in Appendix Table 10 allows a query to be watched against a value (which can be a fixed value or can be a scalar SQL query) and an action to be taken if the criteria match.

4.2.2 Jobs (Appendix Table 3)

Jobs are at the core of AIF because a job is the unit of action that the Messaging Processing Engine (MPE) executes for action_type=run_app. A job definition has 5 sections: metadata, pre-run, run, post_run, and schedule. It is important to note that XML should be provided as an option for input *and* output format so that jobs can be chained**.**

4.2.3 Metadata

Contacts (Appendix Table 5):
This section defines contacts for an Internet application by role such as developer, webmaster, proponent, and operator. The information should be cached/de-normalized for quick lookup.

About (Appendix Table 4):
This XML declaration should match what a component or executable returns as a property or command-line switch. It contains information about the organization, module, versions, last update, and internal/external dependencies.

4.3 Analysis Reports

There is a Reports folder under each log folder. The Reports folder has a Template subfolder. The goal is to create reports and templates for every log. The business value cannot be seen without meaningful and accurate reports.

4.4 Optimization

Well-designed, accurate, and timely analysis reports including baseline reports are required for business decision making and system optimization. Analysis reports/Optimization to business is like what fruits are to trees.

Top

5 External Application Management

5.1 Changes (Appendix Tables 6 & 7)

Change management is a critical component of operation management. One of the first places to check when troubleshooting a problem is recent change. To be effective, we need

1. the list of changes to be up to date.
2. a front-end tool, app_change_UI.htm.
3. a query tool, app_change_query.htm, with client-side scripting and XML capabilities.

In the course of troubleshooting a problem, an analyst *may* make changes that never make it to the corporate change management system. Having this information centralized and easily accessible is very important.

Another important factor is whether a change violates an existing change policy. A change policy may dictate restrictions on file/folder size, type of allowed files, or even based on file content. If you can't enforce a policy, you must at least know when it's violated and take proper action. A QA script, app_change_verifier.js, in the same directory can be saved and registered with a scheduled job.

Application Change Policy sample declaration is in Appendix Table 7.
Application Changes sample declaration is in Appendix Table 6.

5.2 Dependencies (Appendix Table 8)

An XML file based on this fragment in Table 8 keeps track of all dependencies external to the application. For each external resource, there is a predefined test job and optionally an associated contact group to notify in case of failure. The *notification_rules* section allows a group to be contacted if a combination of dependency tests fails. External resources for an Internet application include database, COM component, web service, UNC file, and URL link

An application health-check tool can read this declarative specification of dependencies and perform tests for each on a scheduled basis or an on-demand basis as a troubleshooting tool.

The ultimate goal in this area is for an application management tool to be able to be able to create an Application Dependency Map and behave similarly to what HP Open View does with network devices where if a downstream node becomes unavailable all upstream dependent nodes turn red.

Top

6 Praxis Taxonomy

The following attempts to provide an explanation of how designers of Internet applications can effectively and efficiently turn data into business intelligence.

6.1 Avoid Application Irony

Application Irony is when the utilities or tools for providing the business need for which a software application is created are either weak or non-existent. Application Irony may also be exposed by implementation of a non-AI software product whereby a vast investment is made in the product but follow-up business procedures or practices lack to the extent that the business benefit is only partially realized.

6.2 Business Intelligence Reporting

Reports of reports may be needed and repeated until directly linked with business objectives. Report layers should include:

1. Logs/Data
2. Analysis reports
3. Summary / aggregate reports ➔ Alerts
4. Queries / Views front-end against any of the above items.
5. Dashboard, chart, or pivot table presentation

Alerts are a natural requirement for reports that are typically handled by people but should be automated when comparing results against known business plan goals or past performance.

6.3 Usage Intelligence

This area covers acquiring information and applying it to enhance the user experience. Examples include:

1. Tracking user queries –to create more efficient or cached ones.
2. Analysis of user searches until closure using Successive Query Analysis [4] to give hints about improving application layout or enhancing the search engine.
3. Tracking user personalization choices.

6.4 Business Synonyms

The thesaurus is great for spoken languages but does not cover business languages or technical jargon. The following phrases could be considered equivalent for an IT professional: security, access denied, logon required. Building a proprietary Business Dictionary is very important for effective searches. The issue of Closure mentioned in Usage Intelligence above could work very well with this notion for future searches. The “also-bought” e-commerce approach is also very important to Application Intelligence.

6.5 Value Added Tools (VATs)

Value Added Tools are tools and utilities used to bring an application closer to its business objective or make its management more efficient. They include automation, quality assurance, reporting, or modeling functionality that enhances the service beyond the core requirement. Examples include Policy Compliance Checkers and Log-database upload and query tools.

Top

6.6 Data Dump & Stack Dump

Error logging has gotten more sophisticated and useful [5] –i.e. stack dumps in Java/.NET- but still does not capture say the value of a name such as O'Brien when trying to insert it into a database field using a stored procedure that does not handle the apostrophe in the name. It is extremely helpful to dump important data when reporting errors. An error message shown to the user would have a unique identifier that has a cross reference in a log where the data dump is.

6.7 Database and File System Integration

Databases and file systems are the two main data repositories. Efficient and reliable automation requires trigger support in both in addition to good integration between the two. Triggers work great within database systems but not equally within the file system or between the two. Windows Management Instrumentation, based on the WBEM standard, does have support for folder based triggers but –at this time- with limitations. All what we should have to do –as an example on the Windows platform- is throw a batch file called on_change.bat, on_add.bat, and/or on_delete.bat in any folder and let the operating system take care of the rest. For now, polling or limited application of WMI folder triggers could be used. This impacts the functionality of the InBox and OutBox folders in the proposed folder structure.

6.8 Rules Enforcement

The best implementation of policy/rules is to enforce them. If that is not possible then a policy compliance system must be created where violations are consistently detected, alerted and escalated if need be.

6.9 Documentation

Code files including scripts and program files should follow a standard way of representing help and metadata that would enable

1. retrieval of an About feature as shown in Appendix Table 4
2. a Documentation Engine to:
    a. provide command-line usage help
    b. add code library information
    c. construct internal and external dependencies into an Application Dependency Map (ADM)

Standard file naming should include a ".metadata.XML" suffix. I.e. MyScript.js.metadata.XML which could have a corresponding MyScript.js.metadata.HTML.xsl or MyScript.js.metadata.pdf.xsl.

This expands on the Visual Studio .NET C# capability of turning method signature comments into intell-sense help. The goal is to centralize application help and metadata in an auto-documenting system.

Top

6.10 XML Development Services Infrastructure (XDSI)

Since XML and web services are a mainstream direction for building Internet software functionality, it makes sense to build a development services infrastructure around them. Web services produce XML fragments as return values but what is missing (not just from web services) is a standard way to tell callers whether or not the overall result is a success or failure. This is important for application health-check tools that do not really care about the actual result but rather the overall result. We can standardize on an exact XML tag to be included in *every* web service output such as *overall_result* with values being *success* or *failure*. For existing web services, though, we would still have to parse the output on a case by case basis looking for keywords like "error."

XDSI services ride on whatever XML API in use and are used by business developers. Here are just a few examples of what XDSI could/should include:

1. Standardizing return values to be or include XML fragments –where it makes sense (it doesn't make sense for a Boolean function for instance).

   If other output is mixed with the XML fragment return string, we would have to delimit the fragment in an exact way such as enclosing it with a root node like `<XML_return_value>` and extract it by regular expressions or by parsing

2. Standardize on providing features for
   a. XPATH/XSLT filtering for both input and output, and
   b. XSL formatting for output.

3. Program output options should be:
   a. XML (default)
   b. HTML
   c. ASCII-Converted-HTML (ACH) [4]
      like copying the contents of the browser page into Notepad.
   d. ASCII-Translated-XML (ATX) [4]
      Indented ASCII output with attributes prefixed differently from elements.
      Example:

```
<student id="5">
    <name>Joe Smith</name>
    <address type="home">
        <street>123 main str.</street>
    </address>
</student>
```

      becomes:

```
student
    a: id=5
    e: name=Joe Smith
    e: address
            a: type=home
            e: street=123 main str.
```

Top

4. Standardize program XML input as an option. By enabling programs to produce XML output and accepting it as input, programs can be chained and more consistently automated.

5. Standardize file naming by appending a suffix to the script/program name. Example:
   Input:
   ```
   MyScript.vbs.1.in.XML
   MyScript.vbs.1.in.xslt or MyScript.vbs.1.in.xpath
   ```

   Output:
   ```
   MyScript.vbs.1.out.XML
   MyScript.vbs.1.out.xslt or MyScript.vbs.1.out.xpath
   MyScript.vbs.1.out.HTML.xsl or MyScript.vbs.1.out.pdf.xsl
   or
   MyScript.vbs.1.out.ACH
   MyScript.vbs.1.out.ATX
   MyScript.vbs.1.out.HTML
   ```

Top

7 Conclusion

As this paper has suggested and shown, an intelligent application is one that continues to deliver the business benefit in an efficient way. Having an application that serves business users as well as administrators is indeed a challenge. Application designers need to plan for monitoring application health as much as they need to build whatever infrastructure necessary to provide business reports, summaries, and quality assurance tools. A major component of this infrastructure is a messaging system that allows an application to report runtime errors, performance data, and other data that would help optimize the operation, management, or efficiency of the application. This collected data is meaningless unless 1) it is checked against a baseline and an alert/escalation system is built on top of it 2) the data is analyzed and a query mechanism is provided, and 3) a pleasant presentation output is provided.

Implications for future research may include a challenge to designers and researchers to focus on how applications can become solutions to real-world business issues by providing a standard and/or evaluation system for complete infrastructure, facilities, tools, and reports to realize business benefits.

Top

References

[1] Daniel M. Russell, Paul P. Maglio, Rowan Dordick, Chalapathy Neti (2003), **Dealing with ghosts: managing the user experience of autonomic computing**, March, 2003.

[2] T.K. Landauer, **The Trouble with Computers**, MIT Press, Cambridge, MA (1995)

[3] Liliana Ardissono, Alexander Felfernig, Gerhard Friedrich, Anna Goy, Dietmar Jannach, Giovanna Petrone, Ralph Schafer, Markus Zanker, **A framework for the development of personalized, distributed web-based configuration systems**. AI Magazine, Fall, 2003, by

[4] Nema, Waleed. **Integrated Business Solutions Design**, Training Seminars, 2000-2003.

[5] Justin Murray, **Enhancing application manageability**, June, 2003.

[6] Microsoft Corporation, Microsoft Dynamic Systems Initiative Overview, March 2004.

Top

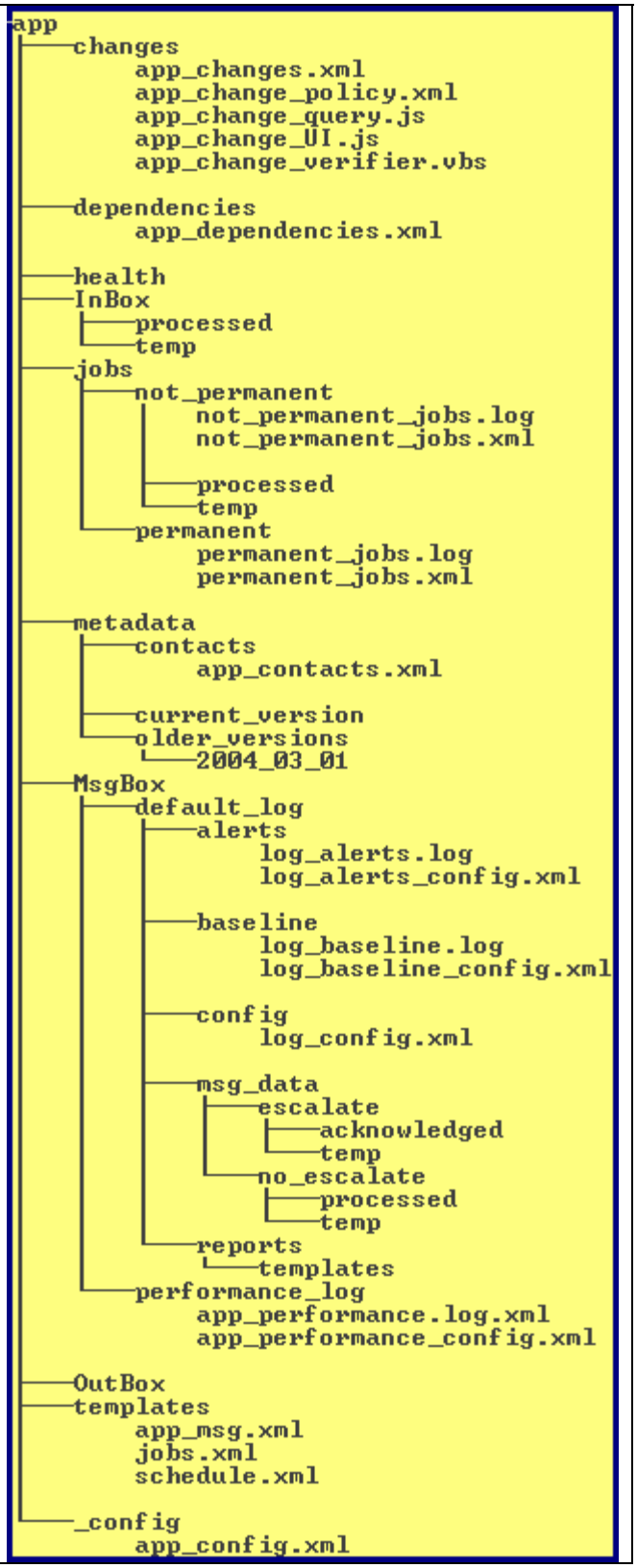
List of Figures and Tables

Figures:

Tables –all in the appendix:

Top

Table 1: AIF Folder Structure

Folder Descriptions	
Changes • Summary of all changes • Change query tool • Change restriction policy • Change compliance checker	
Dependencies A list of all external dependencies.	
Health A specification for health-check scripts	
InBox Application picks messages from here	
Jobs Predefined jobs, scheduled and non-scheduled.	
Metadata Keeps information about the application including contacts and version information	
MsgBox Contains all app logging and messaging. Default log contains: • Alerts • Baseline • Config • Msg_Data Application creates a message here; MsgEngine acts on it. Messages support actions and escalation • Reports	
OutBox Application drops messages here –for pickup by other processes	
Templates Contains schema samples and application relevant templates	
_Config A must for every app, this folder contains all application configuration parameters.	

Top

Table 2: Application Message

```
<app_msgs>
   <app_msg msg_id="">
      <msg_source>
         <source_computer></source_computer>
         <logged_user></logged_user>
         <msg_id></msg_id><!--in case a msg creates another; helps find chain-->
         <source_app>
            <app_id></app_id>
         </source_app>
         <key-value_XML></key-value_XML>
      </msg_source>

      <msg>
         <date></date>
         <time></time>
         <subject></subject>
         <description></description>
         <category></category>
         <event_id></event_id>
         <suggested_action></suggested_action>
         <stack_dump>at least module name where error occurred</stack_dump>
         <data_dump><var>value</var></data_dump>
         <eventlog_type value="0">
               0=success | 1=error | 2=warning |
               4=information | 8=audit success | 16=audit failure
         </eventlog_type>
         <key-value_XML></key-value_XML>
         <escalate acknowledged="no"/>
         <urgency>high | normal | low</urgency>
      </msg>

      <actions><!--multiple actions are allowed-->
         <action type="log" verbose="yes | no">
            <log type="event_log | ASCII | XML | database">
               <!--specific info about each type such as
                  ASCII_delimiter="comma | tab " connection_string="" unc_file_name=""
               -->
            </log>
            <schedule>
               <!--same as /templates/schedule.XML-->
            </schedule>
         </action>

         <action type="email">
            <target name="" type="user | exchnage_group | AD_group | LDAP_group |
own_group"/>
            <schedule>
               <!--same as /templates/schedule.XML-->
            </schedule>
         </action>

         <action type="popup"><!--net send-->
            <target name="" type="machine | user | AD_group | own_group"/>
            <schedule>
               <!--same as /templates/schedule.XML-->
            </schedule>
         </action>

         <action type="run_app"><!--either job_cmd or job_url-->
            <!--same as /templates/jobs.XML-->
            <schedule>
               <!--same as /templates/schedule.XML-->
            </schedule>
         </action>
      </actions>

   </app_msg>
</app_msgs>
```

Top

Table 3: Jobs

```
<jobs>
   <job job_id="" output="XML | HTML | string | numerical_string | none">
      <metadata>
         <title></title>
         <author></author>
         <desctiption></desctiption>
         <contacts_XML></contacts_XML>
         <update></update>
      </metadata>

      <pre_run>
         <if_file_exists file_name="" on_success="continue | abort"
                         on_failure="continue | abort"/>
         <!--existing job(s)-->
         <job job_id="" collect_output="no"
                     on_success="continue | abort" on_failure="continue | abort"
                     on_condition="continue | abort" if_result_xpath="" equals=""/>
         <!--inline job(s) (defined here)-->
         <job_def run_type="cmd | url" collect_output="no"
                     on_success="continue | abort" on_failure="continue | abort"
                     on_condition="continue | abort" if_result_xpath="" equals="">
            <cmd></cmd>
            <run_as user_id="" password=""/>

            <input>
               <cmd_params></cmd_params>
               <!--or-->
               <XML_inline></XML_inline>
               <!--or-->
               <XML_external src=""/>
            </input>
         </job_def>
      </pre_run>

      <run>
         <!--either call an existing job or more to form a batch without dependencies.
             For a batch with dependencies, use pre_run/post_run sections.
         -->
         <job job_id="" collect_output="yes"/>

         <!--or define an inline job. This is THE core unit-->
         <job_def run_type="cmd | url" max_run_milliseconds="" collect_output="yes">
            <cmd></cmd>
            <run_as user_id="" password=""/>

            <input>
               <cmd_params></cmd_params>
               <!--or-->
               <XML_inline></XML_inline>
               <!--or-->
               <XML_external src=""></XML_external>
            </input>
```

Top

```
            <output return="XML | HTML | string | numerical_string | none">
               <success>
                  <by_exit_code if_equals=""/>
                  <!--or-->
                  <by_output>
                     <if>
                        <xpath></xpath>
                        <equals></equals>
                     </if>
                  </by_output>
                  <phrase><!--success phrase-->
                     <content type="output_as_is"/>
                     <!--or-->
                     <content type="output_transformed" xslt_file_name=""/>
                     <!--or-->
                     <content type="literal" format="ASCII | xHTML">content goes
                           here...</content>
                  </phrase>
               </success>
               <failure>
                  <!--same as success section above. provide failure phrase at least-->
                  <phrase><!--failure phrase-->
                     <content type="output_as_is"/>
                     <!--or-->
                     <content type="output_transformed" xslt_file_name=""/>
                     <!--or-->
                     <content type="literal" format="ASCII | xHTML">content goes
                              here...</content>
                  </phrase>
               </failure>
            </output>
         </job_def>
      </run>

      <post_run>
         <on_finish     job_id="" collect_output="no" spawn_no_wait="yes | no"/>
         <on_success    job_id="" collect_output="no" spawn_no_wait="yes | no"/>
         <on_failure    job_id="" collect_output="no" spawn_no_wait="yes | no"/>
         <on_condition  job_id="" if_result_xpath="" equals="" collect_output="no"/>
      </post_run>

      <schedule>
         <run_count remaining_runs="-1" remaining_attempts="-1" max_attempts=""/>
            <!--remaining_attempts=-1 <==> try indefinitely; remaining_runs=-1 <==>
                  permanent task-->
         <date start="" expires_after="" expires_on=""/>
         <time start="" expires_after="" expires_on=""/>
         <cycle>
            <relative_time sleep_seconds="90"/>
            <!--or-->
            <exact_time cycle_seconds="3600" max_run_seconds="3599"/>
            <!--runs on the hour, better for monitoring impact on other system resources-
            ->
         </cycle>
      </schedule>
   </job>
</jobs>
```

Top

Table 4: Metadata Version Information

```
<about>
   <org>
      <org_namespace></org_namespace>
      <org_name></org_name>
      <contacts>
         <contact emp_id="" role="" name="" email="" mobile_phone="" work_phone=""
                  mgr_id=""/>
      </contacts>
   </org>

   <module>
      <name></name>
      <prog_id></prog_id>
      <class_id></class_id>
   </module>

   <versions>
      <this></this>
      <backup></backup>
   </versions>

   <updates>
      <author emp_id="" name="" email="" mobile_phone="" work_phone="" mgr_id=""/>
      <update>
         <date_time></date_time>
         <by emp_id="" name="" email="" mobile_phone="" work_phone="" mgr_id=""/>
         <summary></summary>
      </update>
   </updates>

   <dependencies>
      <internal></internal>
      <external></external>
   </dependencies>
</about>
```

Table 5: Metadata Contacts

```
<app_contacts>
   <contact role="proponent | webmaster | developer | operator">
      <groups>
         <group group_email_id="" manager_id=""
                type="AD_distribution | AD_security | app_group"/>
      </groups>

      <!--resultant set of members for this role (de-normalized)-->
      <members>
         <member emp_id="" email="" work_phone="" mobile_phone="" manager_id=""/>
      </members>
   </contact>

   <app_groups>
      <app_group app_group_id="">
         <group_info>
            <name></name>
            <description></description>
         </group_info>
         <members>
            <member emp_id="" email="" work_phone="" mobile_phone="" manager_id=""/>
            <member emp_id="" email="" work_phone="" mobile_phone="" manager_id=""/>
         </members>
      </app_group>
   </app_groups>

   <cross-ref><!--i.e. list of managers (for escalation, etc.)-->
      <member emp_id="" email="" work_phone="" mobile_phone="" manager_id=""/>
   </cross-ref>
</app_contacts>
```

Top

Table 6: Application Changes

```
<app_changes>
   <change>
      <date></date>
      <time></time>
      <change_management change_request_number="">...</change_management>
      <implementor name="" network_id="" email="" work_phone="" mobile_phone="">
         <notes></notes>
      </implementor>
      <summary>...</summary>
      <detail>...</detail>
      <post_change_QA>
         <sign_off_list>
            <signee title="" name="" network_id="" email="" work_phone=""
                                mobile_phone="">
               <notes></notes>
            </signee>
         </sign_off_list>
      </post_change_QA>
   </change>
</app_changes>
```

Table 7: Application Change Policy

```
<app_change_policy>
   <restrictions>
      <restriction object="file | folder" spec="f*" >
         <size>
            <max spec="any" size_KB="10,000"/>
            <actions_if_violated>
               <!--same section as message actions: log, email, popup, run_app-->
            </actions_if_violated>
         </size>

         <file_types>
            <disallowed app="Microsoft Access" version="any" extension="mdb"/>
            <actions_if_violated/>
         </file_types>

         <content>
            <disallowed if_reg_ex="" matches=""/>
            <actions_if_violated/>
         </content>
      </restriction>
   </restrictions>
</app_change_policy>
```

Top

Table 8: Application Dependencies

```
<app_dependencies>
   <dep_test batch_test_job_id="">
      <!--dep_id's are binary to be additive-->
      <dep dep_id="1" type="db | component | web_service
                            | file | link"
           test_job_id="">
         <description></description>
         <resource>connection_string | ProgId | url | filename</resource>
         <!--every external resource may have its own contact group-->
         <msg_contact></msg_contact>
      </dep>
   </dep_test>

   <notification_rules>
      <if dep_binary="">
         <on_failure>
            <!--app_msg goes here-->
            <msg_contact></msg_contact>
         </on_failure>
      </if>
   </notification_rules>
</app_dependencies>
```

Table 9: Application Performance Config

```
<app_performance_config>
   <counter is_sla_item="yes | no">
      <sampling_frequency></sampling_frequency>
   </counter>

   <flush time_seconds="" size_KB="">
   </flush>
</app_performance_config>
```

Top

Table 10: Log Alerts Config

```
<log_alerts_config>
   <alert>
      <when>sql statement</when>
      <operator>greater than</operator>
      <value>
         <literal></literal>
         <!--or-->
         <baseline>
            <data_source></data_source>
            <sql></sql>
         </baseline>
      </value>
      <actions>
         <!--same section as message actions: log, email, popup, run_app-->
      </actions>
   </alert>
</log_alerts_config>
```

Table 11: Log Baseline Config

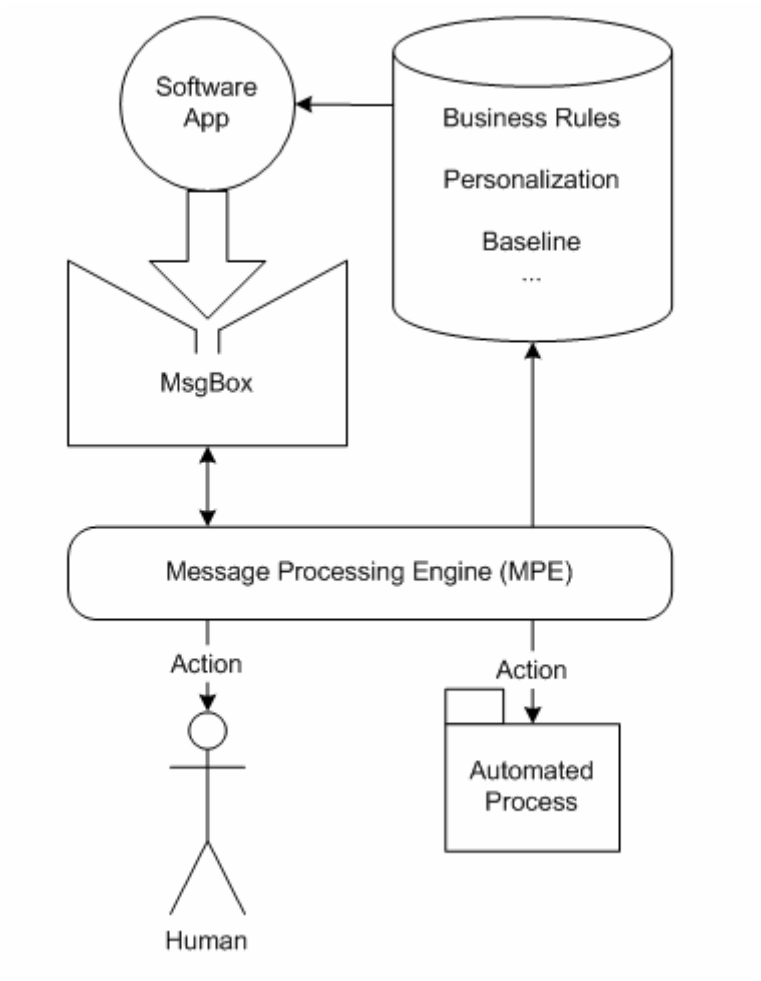
```
<log_baseline_config>
   <sample>
      <description></description>
      <data_set>
         <by_size records=""/>
         <by_time>...</by_time>
      </data_set>
      <field_list>
         <field name="">
            <measure>min</measure>
            <measure>max</measure>
            <measure>median</measure>
            <measure>standard deviation</measure>
         </field>
         <calculated_field>...</calculated_field>
      </field_list>
   </sample>
</log_baseline_config>
```

Table 12: Log Config

```
<log_config id="normal">
   <pre_delim/>
   <post_delim value=",">comma</post_delim>
   <post_delim value="&#13;">Carriage Return/end of line</post_delim>
   <log_duration_hours>24</log_duration_hours> <!--new log every day-->
   <log_naming>yyyymmdd_HH:mm_dow</log_naming>
   <field_list>
      <field name="" data_type="" length="" nullable="1"/>
      <field name="" data_type="" length="" nullable="0"/>
      <field name="" data_type="" length="" nullable="0"/>
   </field_list>
</log_config>
```

Top

Figure 3: Message Processing Engine



Top