# Table of Contents

1. Basics 3
2. Structure 4
   2.1. SYSTEM ................................................................................................................................. 4
      2.1.1. Use case model. .................................................................................................................. 4
      2.1.2. Structure model (dataflow diagrams): ............................................................................... 8
      2.1.3. ASAP system. ..................................................................................................................... 9
      2.1.4. User WEB Interface. ......................................................................................................... 10
      2.1.5. DISPATCHER ..................................................................................................................... 11
      2.1.6. MANAGER. ......................................................................................................................... 12
      2.1.7. CONFIG Generator. .......................................................................................................... 13
      2.1.8. QUERY module. ............................................................................................................... 14
      2.1.9. FINALIZATION module. ................................................................................................. 15
      2.1.10. Receive Answer use case details: .................................................................................... 16
      2.1.11. Collaboration diagram of Receive Answer use case .................................................................. 17
      2.1.12. Sequence diagram of Receive Answer use case. ................................................................ 18
      2.1.13. State/Activity diagrams: ................................................................................................ 19
      2.1.14. Manager ............................................................................................................................ 19
      2.1.15. Ask Question .................................................................................................................... 20
      2.1.16. Receive Answer ................................................................................................................ 21
      2.1.17. Diagram of receive answer with swimlines ...................................................................... 22
      2.1.18. Diagram of receive answer with swimlines ...................................................................... 24
      2.1.19. Diagram of receive answer with swimlines ...................................................................... 24
      2.1.20. Diagram of receive answer with swimlines ...................................................................... 24
      2.2. DATABASE .......................................................................................................................... 25
      2.2.1. System. ............................................................................................................................... 25
      2.2.2. Cache .................................................................................................................................. 32
      2.3. LANGUAGE ........................................................................................................................... 33
      2.3.1. Overview ............................................................................................................................ 33
      2.3.2. Atoms ................................................................................................................................... 33
      2.3.3. Operators ............................................................................................................................. 34
      2.3.4. Initializations, remarks and array accessing ......................................................................... 34
      2.3.5. Statements, blocks and controlling commands ................................................................... 35
      2.3.6. Functions ............................................................................................................................ 36
      Special features: .......................................................................................................................... 37
      2.3.7. Plan script ............................................................................................................................ 37
      2.3.8. Query information .............................................................................................................. 39
      2.3.9. Result script ....................................................................................................................... 39
      2.3.10. Finalization script .............................................................................................................. 43
      2.3.11. Input scripts and query information, Initiate searching. .................................................... 43
      3. Code 46
      3.1. USER WEB INTERFACE ......................................................................................................... 46
      3.2. DISPATCHER ........................................................................................................................... 47
      3.3. MANAGER ............................................................................................................................... 49
      3.4. CONFIG GENERATOR ........................................................................................................... 53
      3.5. QUERY MODULE .................................................................................................................... 55
      3.6. RESULT MODULE ................................................................................................................... 57
      3.7. FINALIZATION MODULE ...................................................................................................... 63
      Specials 66
      4.1. ERROR CODES ......................................................................................................................... 66
      Requirements 67
**Basics**

**Automated Sequence Annotation Pipeline (ASAP)** is designed to ease routine investigation of new functional annotations on unknown sequences, such as expressed sequence tags (ESTs), through querying of web-accessible databases. A simple local database is used to store query formats and parameters, as well as information for the parsing of data received from websites.

The system is a mediator between User, who needs to obtain information, and Useful Servers, which store data, that can help in sequence identification and annotation:

![User-System-Servers Diagram](image)

**Figure 1.1 User-System-Servers**

**Question** – a set of data (e.g., organism, a list of ESTs, what results are needed, etc.) that User specifies to receive information.

**Answer** – a file of any format that contains information received by a series of Query-Result steps.

**Query** – a query to a remote internet server.

**Results** – http- received (txt, xml, etc.) files returned in response to the query from the internet server.

Features of the ASAP system:

- ASAP provides a Web User Interface for User to ask Questions;
- Novel scripting language is used to give instructions to the ASAP system to answer Questions (see chapter 2.3. *LANGUAGE*);
- ASAP system stores all needful data in database (see chapter 2.2. *DATABASE*);
1. Structure

1.1. System

1.1.1. Use case model.

Figure 2.1.1 ASAP Use Cases overview
Actors:

Scientist – the actual user of the ASAP system.

Administrator – the person who writes scripts for the different query plans. Administrator of the ASAP system.

HTML Programmer – the person who creates the web interface for access to the system for Scientist.

E-mail system – a system that provides e-mailing services.

Database system – a database manager system (relative).

File system – the file system of the operating system that ASAP is installed under (see Requirements).

Bio Web Server – Internet server that provides any information.

Use cases:

• Ask Question:
  This use case accepts parameters, organizes it and runs the ASAP system to perform task with this data.

  Precondition:
  HTML Programmer has added an appropriate interface for the questioning system.

  Main Flow:
  This use case begins when User connects to the ASAP query web page, which was created by HTML Programmer. User can now:
  a) Specify all necessary data and parameters to submit query
  b) Submit the query
  c) Go to the ASAP status page

  Use case can:
  a) Return a message about incorrect parameters User submitted
  b) Return a message about success submitting the job with all task identification information

• Check Status
  This use case accepts a task ID or the e-mail address of User and returns the status(es) of the task(s).

  Precondition:
  User is listed in the ASAP status page.

  Main Flow:
  This use case begins when User connected to the ASAP status web page. User can now:
  a) Specify task ID or e-mail to receive status about
  b) Submit query
  c) Go to ASAP query page

  Use case can:
  a) Return a message about incorrect parameters user submitted
  b) Return page with information about all task user submitted
• **Receive Answer**
  This use case does all the routines of the query-result steps, according to the plan that was specified, and sends e-mail to User upon completion.

  Precondition:
  Scientist submitted a query through the ASAP query page. ASAP core has been started with specified parameters.

  Main Flow:
  Only internal objects of the system can use this use case. Use case can:
  a) Get plan, query, result, finalization information from database
  b) Fulfill plan script
  c) Fulfill query making routine
  d) Fulfill file processing routine
  e) Fulfill result composing routine
  f) Store in database all errors occurred
  g) Store information about task status
  h) Send e-mail about results to user

• **Receive Schema – not realized**
  This use case is for centralize SCHEMA receiving. Currently this use case is provided by Administrator who looked through e-mails manually. Schema – is a plan that Scientist wants to be realized via ASAP.

• **Receive Errors – not realized**
  This use case is for centralize ERROR receiving. Currently this use case is provided by Administrator who looked through DATABASE manually.

• **Add/Modify/Delete Script**
  This use case is to add, modify or delete (not realized) plan scripts, query/result/finalization information and format information - to, in or from DATABASE.

  Precondition:
  MySQL database system exists in local server and houses the ASAP system DATABASE.

  Main Flow:
  This use case is for Administrator who wants to add/modify plan script or other information to/in database. Administrator can:
  a) Give a plan script file to the ASAP system
  b) Give a plan script and query/result/finalization information to the ASAP system

  Use case can:
  a) Add a plan script to DATABASE if it doesn’t exist
  b) Modify a plan script to DATABASE if it exists
  c) Add query/result/finalization info to DATABASE if it exists in input file and doesn’t exist in DATABASE
  d) Modify query/result/finalization info to DATABASE if it exists in input file and exists in DATABASE
  e) Add/modify format information if it exists in input file
• **Give Format**
  This use case is realized as a part of the **Add/Modify/Delete Script** use case for testing new plan scripts.

  Precondition:
  MySQL database system exists in local server and it has ASAP database in it.

  Main Flow:
  This use case is for Administrator who wants to add/modify format information to/in database.
  Administrator can:
  a) Add/modify format information passed with plan script by input file

• **Test Script**
  This use case uses the **Receive Answer** use case.

  Precondition: None

  Main Flow:
  This use case is for Administrator who wants to test a plan script. Administrator can:
  a) Pass a file with question parameters and a plan script (and query/result/finalization optional) as an input to run ASAP core
  b) Get debugging information from standard output

  Use case can (apart of **Receive Answer** use case abilities):
  a) Write debugging information to standard output

• **Receive Format – not realized**
  This use case is for centralize FORMAT receiving. Currently this use case is provided by HTML Programmer who looked through DATABASE manually to get information about Format. Format – is a set of variables with definitions to write the web interface, which is provided for a particular plan, properly.

• **Add/Modify/Delete HTML – not realized**
  This use case is for centralize storing of HTML pages and CGI scripts. Currently this use case is provided by HTML Programmer who adds/modifies/deletes files in the file system manually.

• **Receive Query**
  This use case is realized as a part of **Receive Answer** use case for querying Internet servers.

  Precondition:
  File with query parameters specified has been passed as an input.

  Main Flow:
  Only internal objects of the system can use this use case. Use case can:
  a) Fulfill query making routine
  b) Redo query making routine in case of http redirecting
  c) Store in database all errors occurred
• **Give result**
  This use case is realized as a part of the **Receive Answer** use case for taking results from Internet servers as a response to a query.

  Precondition:
  **Receive Query** use case just finished its processing.

  Main Flow:
  Only internal objects of the system can use this use case. Use case can:
  a) Fulfill file processing routine
  b) Fulfill **Receive Query** use case in case of redirecting by html header
  c) Store in database all errors occurred

*Structure model (dataflow diagrams):*

The next section of SYSTEM documentation is Structure model of ASAP system.

There are dataflow diagrams for all modules of code. Notation for those diagrams:

- Solid rectangle border & *.pl caption – constant Perl script
- Solid rectangle border & not *.pl caption – constant module or logic part of module or file
- Solid rectangle border inside DB – database table
- Solid rhombus border – constant file
- Dotted rectangle border – dynamically generated Perl script (temporary)
- Dotted rhombus border – dynamically generated file (temporary)
- Arrows and captions – direction and information of data flowing
1.1.2. ASAP system.

Figure 2.1.2. ASAP system modules
1.1.3. User WEB Interface.

Figure 2.1.3. User WEB interface module
1.1.4. DISPATCHER.
1.1.5. MANAGER.

Figure 2.1.5. MANAGER module
1.1.6. CONFIG Generator.

Figure 2.1.6. CONFIG generator module.
1.1.7. QUERY module.
1.1.8. RESULTS module.
1.1.9. FINALIZATION module.

Figure 2.1.9. FINALIZATION module
Receive Answer use case details:
This section provides some behavior details of the Receive Answer use case – see Figures 2.2.10 and 2.2.11 (see also 2.1.15. and 2.1.16.)


Figure 2.1.10 Collaboration diagram of Receive Answer use case

The next sequence diagram shows, in advance of collaboration diagram, some information about run time between different events or messages:
1.1.1. Sequence diagram of Receive Answer use case.

Figure 2.1.11 Sequence diagram of Receive Answer use case.
State/Activity diagrams:

1.1.12. Manager

Figure 2.1.12 State diagram of Manager module.
1.1.13. Config Generator

Figure 2.1.13 State diagram of Config generator module.
1.1.14. Ask Question

Figure 2.1.14 Activity diagram of Ask Question use case.
1.1.15. Receive Answer

Figure 2.1.15 Activity diagram of Receive answer use case.
1.1.16. Diagram of receive answer with swimlines.

Figure 2.1.16 Activity diagram of Receive answer use case with swimlines
1.2. Database

1.2.1. System.

ASAP system uses MySQL database system. Name of the database - ASAP

Figure 2.2.1 – is an ER-diagram shows tables of database and there’s relations. Tables’ and its fields’ description is provided further:

**CONST_PARAM**
Constant parameter of the query that has a certain value.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_cp</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_query</td>
<td>INT</td>
<td>Identifier for query that has this parameter</td>
<td>NOT NULL</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>cp_name</td>
<td>VARCHAR(50)</td>
<td>Name of the parameter</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>cp_value</td>
<td>TEXT</td>
<td>Value of the parameter</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**ERROR**
Information about an error that has occurred.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_error</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_question</td>
<td>INT</td>
<td>Identifier for question with the error</td>
<td>NOT NULL</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>err_package</td>
<td>VARCHAR(15)</td>
<td>Name of the package with the error</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>err_code</td>
<td>INTEGER</td>
<td>Code of the error</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>err_description</td>
<td>VARCHAR(250)</td>
<td>Description of the error</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>err_details</td>
<td>VARCHAR(50)</td>
<td>Error details</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>err_date</td>
<td>DATETIME</td>
<td>The date when error occurred</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Figure 2.2.1 ER-diagram of ASAP database.
### FINAL
Finalization script

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_final</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>final_name</td>
<td>VARCHAR(15)</td>
<td>Short name for the finalization script (pair name-number is unique)</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>final_numb</td>
<td>INTEGER</td>
<td>Number for the finalization script (pair name-number is unique)</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>final_code</td>
<td>TEXT</td>
<td>Text of the finalization code</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Final_date</td>
<td>DATETIME</td>
<td>Date when the finalization code text has been created</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### KEYWORD
Keyword of the site.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_keyword</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site_keyword</td>
<td>VARCHAR(50)</td>
<td>Special keyword of a site</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### PARAMETER
Format parameter – shows what and where should be pointed in question.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_parameter</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ID_script</td>
<td>INT</td>
<td>Identifier for script this parameter belongs</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>parameter_number</td>
<td>INTEGER</td>
<td>Ordinal number of the parameter in the format</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>parameter_description</td>
<td>VARCHAR(250)</td>
<td>Description of the parameter</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
### QUERY
Information about query to a server.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_query</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Query_name</td>
<td>VARCHAR(15)</td>
<td>Short name for the query (pair name-number is unique)</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Query_numb</td>
<td>INTEGER</td>
<td>Number for the query (pair name-number is unique)</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Query_desc</td>
<td>VARCHAR(250)</td>
<td>Description of the query</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>query_url</td>
<td>VARCHAR(250)</td>
<td>URL of the query</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Query_date</td>
<td>DATETIME</td>
<td>Date when the information on this query has been received</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### QUESTION
Certain question that scientist asked.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_question</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ID_script</td>
<td>INT</td>
<td>Identifier for script that the question belongs to</td>
<td>NOT NULL</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>question_mail</td>
<td>VARCHAR(50)</td>
<td>E-mail of final answer receiver</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>question_date</td>
<td>DATETIME</td>
<td>Date when the question has been asked</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### QUESTION_PARAM
Question parameter – shows what value and where is pointed in question.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_qp</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ID_question</td>
<td>INT</td>
<td>Identifier for question that the parameter belongs to</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>qp_number</td>
<td>INTEGER</td>
<td>Ordinal number of the parameter of the question</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>qp_value</td>
<td>VARCHAR(250)</td>
<td>Value of the parameter</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
**RESULT**
Result script

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_result</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Result_name</td>
<td>VARCHAR(15)</td>
<td>Short name for the result script (pair name-number is unique)</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Result_numb</td>
<td>INTEGER</td>
<td>Number for the result script (pair name-number is unique)</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Result_code</td>
<td>TEXT</td>
<td>Text of the result code</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Result_date</td>
<td>DATETIME</td>
<td>Date when the result code text has been created</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**SCRIPT**
Basic script

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_script</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Script_code</td>
<td>TEXT</td>
<td>Text of the script code</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Script_date</td>
<td>DATETIME</td>
<td>Date when the script code text has been created</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>format_description</td>
<td>VARCHAR(250)</td>
<td>Description of the script format</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>format_date</td>
<td>DATETIME</td>
<td>Date when the format has been affirmed</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**SITE**
Information about a site.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_site</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site_name</td>
<td>VARCHAR(250)</td>
<td>Symbolical site name</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Site_url</td>
<td>VARCHAR(250)</td>
<td>Site url</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>site_description</td>
<td>TEXT</td>
<td>Site description</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
**USER**
User who started task.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_user</td>
<td>VARCHAR(14)</td>
<td>Identifier: yymmddhhmmss**, where <em>random</em></td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>mail</td>
<td>VARCHAR(50)</td>
<td>E-mail of user</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>result</td>
<td>VARCHAR(50)</td>
<td>Result that user need (flag sequence)</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>busy</td>
<td>TINYINT</td>
<td>NULL-task finished, 1-not finished</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**USER_PARAM**
Parameter of the query that has user-defined value.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_up</td>
<td>INT</td>
<td>Identifier</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_query</td>
<td>INT</td>
<td>Identifier for query that has this parameter</td>
<td>NOT NULL</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>up_name</td>
<td>VARCHAR(50)</td>
<td>Name of the parameter</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**has_keyword**
Relation between SITE and KEYWORD

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_site</td>
<td>INT</td>
<td>Identifier for site</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_keyword</td>
<td>INT</td>
<td>Identifier for this site keyword</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**query_of**
Relation between QUERY and SITE

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_query</td>
<td>INT</td>
<td>Identifier for query</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_site</td>
<td>INT</td>
<td>Identifier for site which this query hits</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
**use_final**
Relation between SCRIPT and FINAL

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_script</td>
<td>INT</td>
<td>Identifier for script</td>
<td>NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_final</td>
<td>INT</td>
<td>Identifier for final which this script uses</td>
<td>NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**use_query**
Relation between SCRIPT and QUERY

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_script</td>
<td>INT</td>
<td>Identifier for script</td>
<td>NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_query</td>
<td>INT</td>
<td>Identifier for query which this script uses</td>
<td>NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**use_result**
Relation between SCRIPT and RESULT

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_script</td>
<td>INT</td>
<td>Identifier for script</td>
<td>NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ID_result</td>
<td>INT</td>
<td>Identifier for result which this script uses</td>
<td>NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
1.2.2. Cache.
Cache data stores in database in tables like this:

![Cache Table Template](image)

Figure 2.2.2. Table template

Attributes description:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Data Type</th>
<th>Comment</th>
<th>NULL</th>
<th>PK</th>
<th>FK</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID_number</td>
<td>INT</td>
<td>Identifier for table</td>
<td>NOT NULL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>date</td>
<td>DATETIME</td>
<td>Date and time when caching happened</td>
<td>NOT NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ID</td>
<td>VARCHAR(255)</td>
<td>Custom ID</td>
<td>NULL</td>
<td>No</td>
<td>Yes/No</td>
</tr>
<tr>
<td>c1</td>
<td>TEXT</td>
<td>Custom attribute #1</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>cN</td>
<td>TEXT</td>
<td>Custom attribute #N</td>
<td>NULL</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
1.3. Language

1.3.1. Overview

To make ASAP work you need to pass instructions to it regarding what to do. These instructions are: plan script, query information, result script, finalization script.

*Plan script* – defines the sequence of that actions: hitting servers, processing results and composing reports. See clause 2.3.7 for special details about Plan script.

*Query information* – defines query parameters: remote cgi-script address, parameters to pass, description of the query purpose. See clause 2.3.8 for special details about Query information.

*Result script* – defines handling of the result page received in response to a query. See clause 2.3.9 for special details about Results script.

*Finalization script* – defines handling of input values passed to this script. See clause 2.3.9 for special details about Finalization script.

For script programming a special language has been created as an extension of perl. This script language can operate numbers, strings and arrays of numbers and strings. All operations should be divided by ‘new line’ character. A semi-colon ( ; ) at the end of the string is optional.

1.3.2. Atoms

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol</th>
<th>Example</th>
<th>Is a name for:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>$</td>
<td>2</td>
<td>Number</td>
</tr>
<tr>
<td>String</td>
<td>''</td>
<td>‘string’</td>
<td>String</td>
</tr>
<tr>
<td>Scalar</td>
<td>$</td>
<td>$scalar_name</td>
<td>An individual value (number, string)</td>
</tr>
<tr>
<td>Array</td>
<td>@</td>
<td>@array_name</td>
<td>A list of numbers, keyed by number</td>
</tr>
</tbody>
</table>

*Table 2.3.1 Atoms*
1.3.3. Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoincrement</td>
<td>++</td>
<td>$a++</td>
<td>Sum of $a and 1</td>
</tr>
<tr>
<td>Autodecrement</td>
<td>--</td>
<td>$a--</td>
<td>Difference between $a and 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>$a + b</td>
<td>Sum of $a and $b</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>$a - b</td>
<td>Difference between $a and $b</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>$a * b</td>
<td>Product of $a and $b</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>$a / b</td>
<td>Quotient of $a and $b</td>
</tr>
<tr>
<td>Modulus</td>
<td>%</td>
<td>$a % b</td>
<td>Remainder of $a divided by $b</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>**</td>
<td>$a ** b</td>
<td>$a to the power of $b</td>
</tr>
</tbody>
</table>

**Unary arithmetic operators**

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenation</td>
<td>.</td>
<td>$a . b</td>
<td>Concatenation of $a and $b</td>
</tr>
<tr>
<td>Repetition</td>
<td>*</td>
<td>$a x $b</td>
<td>Repeat string $a $b times</td>
</tr>
</tbody>
</table>

**Binary arithmetic operators**

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>&amp;&amp;</td>
<td>$a &amp;&amp; $b</td>
<td>$a, if $a is false. $b otherwise</td>
</tr>
<tr>
<td>Or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not</td>
<td>!</td>
<td>! $a</td>
<td>True, if $a is not true</td>
</tr>
</tbody>
</table>

**Logical operators**

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td>== or eq</td>
<td>$a = = $b</td>
<td>True if $a is equal to $b</td>
</tr>
<tr>
<td>Not equal</td>
<td>!= or ne</td>
<td>$a != $b</td>
<td>True if $a is not equal to $b</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt; or lt</td>
<td>$a &lt; $b</td>
<td>True if $a is less than $b</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt; or gt</td>
<td>$a &gt; $b</td>
<td>True if $a is greater than $b</td>
</tr>
<tr>
<td>Less than or equal</td>
<td>&lt;= or le</td>
<td>$a &lt;= $b</td>
<td>True if $a not greater than $b</td>
</tr>
<tr>
<td>Greater than or equal</td>
<td>&gt;= or ge</td>
<td>$a &gt;= $b</td>
<td>True if $a not less than $b</td>
</tr>
<tr>
<td>Comparison</td>
<td>&lt;=&gt; or cmp</td>
<td>$a &lt;=&gt; $b</td>
<td>0 if equal, 1 if $a greater, -1 if $b greater</td>
</tr>
</tbody>
</table>

**String operators**

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>=</td>
<td>$a = $b</td>
<td>Assign to $a value of $b</td>
</tr>
<tr>
<td>Overassignment</td>
<td>o=</td>
<td>$a o= $b</td>
<td>Here o – operator and $a = $a o $b</td>
</tr>
</tbody>
</table>

**Comparison operators**

**Table 2.3.2 Operators.**

1.3.4. Initializations, remarks and array accessing.

To initialize variable $var with a number:
$var = 1

To initialize variable $var with a string:
$var = "Any symbols"

To initialize array:
@arr = (“element1”, “element2”)

To make a remark – use ‘#’ or ‘//’ symbols:
# this is a remark
// this is a remark also

To receive any element of array, use [] brackets:
$arr[0] – element of array with index = 0
Notice, that $arr[0] – is a scalar!! So it is ‘$’ symbol ahead.
### 1.3.5. Statements, blocks and controlling commands

**if statement:**
Syntax:
```plaintext
if(condition){
    # condition=TRUE block
}
else{
    # condition=FALSE block
}
```
Example:
```plaintext
if($a==$b){
    $a = "one";
} else{
    $a = "two";
}
```
Remarks:
‘else’ block can be omitted.

**for loop statement:**
Syntax:
```plaintext
for(initialization; condition; re-initialization){
    # condition is still TRUE
}
```
Example:
```plaintext
for($i=0; $i<15; $i++){
    @array[$i] = $i
}
```
Remarks:
Any of initialization, condition or re-initialization can be omitted. Symbol ‘1’ is required in this case ( for(1; 1; 1){ . . . } - infinite loop )

**while loop statement:**
Syntax:
```plaintext
while(condition){
    # condition is still TRUE
}
```
Example:
```plaintext
$i = 0;
while($i<15){
    $i++
}
```
Remarks:
The example does the same as ‘for loop statement’ example. Condition also can be omitted by replacing with ‘1’.

**block:**
Syntax:
```plaintext
NAME: {
    # block code
}
```
Example:
```
MY_BLOCK: {
    last MY_BLOCK
}
```

Remarks:
The block semantically equivalent to a loop that executes once. You can use block to mark pieces of code to use `last` command.

**last controlling command:**
Syntax:
```
last NAME
```
Example:
```
MY_BLOCK: {
    last MY_BLOCK
}
```
Remarks:
Use this command to finish the block execution.
You can omit `NAME` – then script finish execution of current loop statement.

**exit controlling command:**
Syntax:
```
exit
```
Example:
```
if($error){
    exit
}
```
Remarks:
Use this command to finish execution of WHOLE script

### 1.3.6. Functions

**sleep:**
Syntax:
```
sleep ($number)
```
Example:
```
sleep(15)
```
Description:
Execution of the script pauses for `$number` seconds

**print:**
Syntax:
```
print ($scalar)
```
Example:
```
print ("Any string you want!
")
```
Description:
Use this function to print to standard output (for debug purposes). Brackets can be omitted.

**length:**
Syntax:
```
length($string)
```
Example:
```
$strlen = length("String")
```
Description:
Use this function to get string length. (In example $strlen = 6)

push:
Syntax:
    push (@array, $scalar)
    push (@array, @another_array)
Example:
    @abab = ()
    push @abab, “a”
    push @abab, “b”
    push @abab, @abab
Description:
Use this function to append to the end of the array scalar or another array. The result of the example is array @abab = (“a”, “b”, “a”, “b”). Brackets can be omitted.

join:
Syntax:
    $string = join ($devider, @array)
Example:
    @abab = (“a”, “b”, “a”, “b”)
    $abab = join (“_”, @abab)
Description:
Use this function to compose all elements of the array into one string – appended one by one elements with $devider between them. The result of the example is string $abab = “a_b_a_b”.

split:
Syntax:
    @array = split ($devider, $string)
Example:
    $abab = “a_b_a_b”
    @abab = split (“_”, $abab)
Description:
Use this function to make an array from the string – elements will be chunks of the strings between string represented by $devider. The result of the example is array @abab = (“a”, “b”, “a”, “b”).

Special features:

1.3.7. Plan script

There are additional special functions that were defined for plan script: query, result, final, and db_get, db_put, db_del for caching. Plan script can receive input parameters from question of user. You can use this parameters by accessing to it by its number (1-based) like this: <!--1-->, <!--2-->, etc. (e.g $my_var = <!--1-->). See also clause 2.3.11 to figure out ways to include plan script to ASAP system.

query:
Syntax:
    query($ID_NAME, $ID_NUMBER, $parameter1, ..., $parameterN)
Example:
    query(TIGR, 1, “AA598665”, 3)
Description:
Use this function to make a query to the server. Pair $ID_NAME - $ID_NUMBER is unique for every query information that will be taken to make a query. $parameter1, ..., $parameterN will be used to specify query. See clause 2.3.8 for special details about query information.

result:
Syntax:
result($ID_NAME, $ID_NUMBER, $parameter1, ..., $parameterN)
Example:
result (TIGR, 1, $tc_id, @tigr_GO, $tigr_ANN)
Description:
Use this function to call result script to process result page that was received as a response to the last query that was specified by last query function. Pair $ID_NAME - $ID_NUMBER is unique for every result script that will be taken for processing. $parameter1, ..., $parameterN will be used as input values to the script or to return pulled from the page information (depends of the internals of the script). In the example: script with identification pair (TIGR, 1) processes page and return to the variables $tc_id – ID of theoretical contig, @tigr_GO – array of GO numbers and $tigr_ANN – annotation of the EST. See clause 2.3.9 for special details about result script.

final:
Syntax:
final($ID_NAME, $ID_NUMBER, $parameter1, ..., $parameterN)
Example:
final(FILE, 1, “my_file.txt”, “AA598665”, $tigr_ANN)
Description:
Use this function to call finalization script to put some information to the file or to compose information. Pair $ID_NAME - $ID_NUMBER is unique for every finalization script that will be taken for processing. $parameter1, ..., $parameterN will be used as input values to the script or to return pulled from the page information (depends of the internals of the script). See clause 2.3.10 for special details about finalization script.

db_get:
Syntax:
db_get($TABLE_NAME, $ID, @array)
Example:
$entries_found = db_get(“go”, “GO:0015644”, @put_GO_here)
Description:
Use this function to check cache for the presence of entry. Provide $ID as a identification string of the entry. This function puts output information to the array provided as the third parameter of the function. It is two-dimensional array: every row – is different entry found, column is different information about this entry. First column (index 0) – always identification string you passed to the function ($ID). Function returns number of entries found in cache. NOTICE, that before array symbol ‘@’ should be backslash symbol!!!

db_put:
 Syntax:
db_put($TABLE_NAME, $ID, @array)
Example:
@GO_to_cache = (1, “GO:0015643 – anti-toxin\tGO:0015644 – lipoprotein anti-toxin”) db_put(“go”, “GO:0015644”, @GO_to_cache)
Description:
Use this function to cache some information. Provide $ID as an identification string of the entry. This function takes information of the entry from the array provided as the third parameter of the function. It is one-dimensional array: every element is different information about this entry.

**db_del**:

Syntax:
```
db_del($TABLE_NAME, $ID)
```

Example:
```
db_del("go", "GO:0015644")
```

Description:
Use this function to delete from cache some entry. Provide $ID as an identification string of the entry to delete. Use this function before you want to update cache with db_put function. Otherwise it will be two entries with the same $ID in the cache (if one existed before).

### 1.3.8. Query information

Query information is provided as fielded tab-delimited plain data. This information consists of three strings: address, parameters and description:

```
[address]<tab>http://full_path_to_the_server_script
[parameters]<tab>param1_name=param1_value<tab>...<tab>paramN_name=paramN_value
```

Here `<tab>` - is tabulation symbol.

You can use input parameters from plan script by its numbers (1-based) like this: `<!--1-->`, `<!--2-->`, etc. (e.g param1_name=<!--1-->). Number of parameter is defined by its position in plan script query function - `query($ID_NAME, $ID_NUMBER, $parameter1, ..., $parameterN)`

Example:
```
[address]<tab>http://www.tigr.org/tigr-scripts/tgi/tc_report.pl
[parameters]<tab>tc=<!--1--><tab>species=<!--2-->
[description]<tab>Query to get tc sequence by its ID.
```

See also clause 2.3.11 to figure out ways to include query information to ASAP system.

### 1.3.9. Result script

There are additional special functions that were defined for result script: ERROR, GET and GET_ALL. Result script can receive input parameters from plan script by its numbers (1-based) like this: `<!--1-->`, `<!--2-->`, etc. (e.g `$my_var=<!--1-->`). Number of parameter is defined by its position in plan script result function - `result($ID_NAME, $ID_NUMBER, $parameter1, ..., $parameterN)`. See also clause 2.3.11 to figure out ways to include result script to ASAP system.

**ERROR**:

Syntax:
```
ERROR($string)
```

Example:
```
ERROR("Error of parsing - maybe page format changed!")
```
Description:
Use this function to finish result script executing with error. The error will be stored in database with code 04 (04 - parsing error (result module)) – see chapter 4.1 for all error codes) and information about current question and error string.

GET:
Syntax:
GET(source, target, condition, output)
Example:
GET(WHOLE, TABLE, name=LIKE(tbl.) & align=center & border=1, @array=TRTD[0,2][0-8])

This function looks through whole page that was received by last query function of plan script. It looks for FIRST <table> tag with attributes align="center", border="1" and name="tbl." where '.' - is any symbol. If the function founds table like this, it gets rows 0 and 2 and columns from 0 to 8 and puts it into array @array. This array will be two dimensional: 2 rows and 9 columns.

Description:
Use this function to parse whole page or some variable content. Function initialize variable $get_result with 1 if target found. 0 otherwise. See next information about format of source, target, condition and output parameters.

source: Source of information (html or plain text) to parse. Can be:
- $string - string buffer to parse.
- 'WHOLE' - whole page that is a result from last query from plan script.

target: Shows what to find in source. Can be:
- tag_name - any HTML tag name (e.g html, meta, table, br, font)
- 'TEXT' - function will search only in text, omitting all tags.

condition: Condition that target should satisfy. Symbols '|' and '&' can be used to make multiple condition. Can be:
- 1 - means any target
- tag_attr=constant - certain target attribute (in case if target is some tag) is equal some constant (e.g value="Human", border=1, size="+1", etc.)
- 'NUMBER'=constant - ordinal number of target that satisfy all other conditions. Number is 1-based. (e.g NUMBER=2 - means second target that function will find)
- 'TEXT'=constant - text of the target (usually - it is text between <tag> and its terminator)
- 'HTXT'=constant - hypertext of the target (usually - it is everything between <tag> and its terminator)

Instead of constant you can use pattern matching by using LIKE(pattern) instead of constant. (e.g TEXT=LIKE(pattern)). If word LIKE is written with all small letters (like) - pattern matching will be case-insensitive. See description of pattern after function definition.
**output:** If `target` that satisfy `condition` is found then this parameter shows where what information we need to store from the `target` and where. '&amp;' can be used to make multiple `output`. Can be:

1 - means that we don’t need anything

`variables=TRTD[range][range]` - get table cells (`target` should be ‘TABLE’!).

`range` defines what table cells to take. `range` - are numbers, separated by ',', or '-', '.' , '=' and, '-', '=' from, to (e.g. 0,3-7). So, for instance `TRTD[0,2][0-3]` - means cells that in rows 0, 2 and columns 0, 1, 2, 3. ‘TRTD’ can be used without one or both [range]. If one [range] omitted - then function will take all columns and rows defined in existed [range]. If both of [range] are omitted - function takes all cells of table into variable

`variables=TEXT[range][range]` - gets text of `target` into variable if both [range] are omitted. If one of it exists - gets only strings with indexes within `range`. If both [range] exist - gets strings within first range and letters within second. E.g if text is “abcdefg” then `TEXT[0][1-3, 5]` will give us an array with two elements: “bcd” and “f”.

`variables=HTXT[range]` - gets hypertext of `target` into variable if [range] is omitted. If it exists - gets only strings with indexes within `range`. If variable is array - function fill it with strings. If scalar - strings are concatenated.

`variables=ALL[range]` - gets all hypertext after `target` (even if it not belongs to the `target`) if [range] is omitted. If it exists - gets only strings with indexes within `range`. If variable is array - function fill it with strings. If scalar - strings are concatenated.

`scalar=tag_attr` - function puts `tag_attr` value of the `target` into scalar

`scalar=NUMBER` - function puts ordinal number of the `target` into scalar

`scalar=CAPTURE[i][j]` - function puts value captured from LIKE `pattern` of the `target` `condition` into scalar (i - number of condition statement (0-based), j - number of capture brackets there (See `pattern` definition for capture details))

**pattern:** Pattern is a symbol sequence with ones that have special meaning:

* - any number of any symbols

+ - at least one any symbol

? - 0 or 1 any symbol

. - 1 any symbol

^ - start of the string

$ - end of the string

| - alternation (match one or the other)

{} - grouping (for '|' scope and *capturing*)

[range] - number satisfied the range ([15-143] - any number between 15 and 143)

[symbol range] - symbol that satisfy the symbol range ([A-Z] - any capital letter)

\ - symbol to make all above non-special
!greed – special marker – should be used after `'*' or `+'. By default: `'*' and `+' means as much as possible (greedy matching). In case of `!greed' using – that will mean as few as possible.

Use pattern in condition:

• when you need to make this condition less strict. E.g. condition TEXT=like(human) matches strings "Human", "Is a human", "Human mouse" and condition TEXT="human" matches only "Human". Or TEXT=like(t*t) matches "test", "tonight" and "tea time", etc.

• when you need to capture something from searching string: use curl brackets {} and CAPTURE in output parameter to do it. E.g.

GET('<font size=+1>x=5</font>', FONT, size=+1 & TEXT=like(x={}), $x_equal=CAPTURE[1][0])

will assign '5' to $x_equal variable. First index of CAPTURE is a 0-based index of the condition part to capture from (it is two condition parts in the example). Second index - is a 0-based index of curl brackets to capture from - indexes of curl brackets are counted by the location of the left parenthesis. So, {{First} {Second}} pattern applying to "First Second" string gives: CAPTURE[0][0]="First Second", CAPTURE[0][1]="First", CAPTURE[0][2]="Second".

Examples of pattern matching:

m?n matches "man", "men", "min" but not "moon".

t*t matches "test", "tonight" and "tea time" (the "tea t" portion)

Te+st matches "test", "teest", "teeeest" etc. but does not match "tst".

^Word$ matches "Word" but not "A word" or "Words"

[a,e,i,o,u] matches every lowercase vowel

[0-9,a-z] matches any digit, or lowercase letter

[15-40] matches any number between 15 and 40.

You may search for an expression A or B as follow:
expressionA|expressionB

This will search for an occurrence of expressionA OR expressionB. Scope of `|' alternation can be pointed by curl brackets. Compare:

x=5|6|7|8=y – matches ’x=5’, ’6’, ’7’ or ’8=y’

x={5|6|7|8}=y – matches ’x=5=y’, ’x=6=y’, ’x=7=y’ or ’x=8=y’

GET_ALL:

Syntax:
GET_ALL(source, target, condition, output)

Example:
GET_ALL(WHOLE, TABLE, name=LIKE(tbl.), @array=TRTD[0,2][0-8])

This function looks through whole page that was received by last query function of plan script. It looks for ALL <table> tags with name="tbl." where '.' – is any symbol. If the function founds table like this, it gets rows 0 and 2 and columns from 0 to 8 and puts it into array @array. This array will be TREE dimensional: first dimension will be index of table found and other two dimensions: 2 rows and 9 columns.
Description:
Use this function to parse whole page or some variable content and get ALL found targets that satisfy condition. This function acts like GET function except of output: it adds extra-dimension to the variable used in output. Function initialize variable $get_result with number of targets found. 0 if not found.

1.3.10. Finalization script
There is additional special function that was defined for finalization script: PUT. Finalization script can receive input parameters from plan script by its numbers (1-based) like this: <!--1-->, <!--2-->, etc. (e.g $my_var=<!--1-->). Number of parameter is defined by its position in plan script final function – final($ID_NAME, $ID_NUMBER, $parameter1, ..., $parameterN). See also clause 2.3.11 to figure out ways to include finalization script to ASAP system.

PUT:
Syntax:
PUT(destination, target, attributes, input)
Example:
PUT("<!--1-->\t<!--2-->\n")

This function writes to file ‘<!--1-->’ string "<!--2-->\t<!--3-->\n"

Description:
Use this function to write to file destination tag or text target that has attributes and information input. It creates file if it doesn’t exist, and append information otherwise. See next information about format of destination, target, attributes and input parameters.

destination: Path to file to put information (html or plain text) to. Can be:
  $string - string with file name.

target: Shows what to find in source. Can be:
  tag_name - any HTML tag name (e.g html, meta, table, br, font)
  ‘TEXT’ - function will write plain text.

  NOTE: only ‘TEXT’ target realized on this moment

attributes: Symbol '&' can be used to point multiple attributes. Can be:
  1 - no attributes
  tag_attr=constant - certain value of certain target attribute

input: Variables to fill target with Can be:
  variable - scalar or array, depends on target (e.g if target is ‘TABLE’ - variable should be array).

1.3.11. Input scripts and query information. Initiate searching.
This clause describes how to include plan script, query information, result script, finalization script and initiate searching process of ASAP system.
There are two ways to initiate searching process of ASAP system with certain plan search:

- have whole set of plan script, query information, result script, finalization script in ASAP database. And pass configuration file to Manager with just Question section in it (see Table 2.3.11 for sections details and chapter 3 for Manager input format).
- pass configuration file to Manager with all sections, so ASAP doesn’t need to look through database to find plan script, query information, result script or finalization script. This way is used for debugging new plan scripts, query information, result scripts or finalization scripts. ID of plan script should be ZERO in Question section.

<table>
<thead>
<tr>
<th>Section</th>
<th>Marker</th>
<th>Content</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question</td>
<td>&lt;QUESTION&gt;</td>
<td>User Question</td>
<td>User question – is a set of initial parameters: each parameter in new string. First parameter – ID of plan script</td>
</tr>
<tr>
<td>Script</td>
<td>&lt;SCRIPT&gt;</td>
<td>Plan Script</td>
<td>Use remarks just after this section marker to set plan script ID and description of parameter set of user question by this pattern:</td>
</tr>
<tr>
<td>Query</td>
<td>&lt;QUERY&gt;</td>
<td>Query Information</td>
<td>To set query information ID use pair {NAME, NUMBER} before each query information.</td>
</tr>
<tr>
<td>Result</td>
<td>&lt;RESULT&gt;</td>
<td>Result Script</td>
<td>To set result script ID use pair {NAME, NUMBER} before each result script.</td>
</tr>
<tr>
<td>Finalization</td>
<td>&lt;FINAL&gt;</td>
<td>Finalization Script</td>
<td>To set finalization script ID use pair {NAME, NUMBER} before each finalization script.</td>
</tr>
</tbody>
</table>

Table 2.3.11 Configuration file.

Since configuration file with find plan script, query information, result scripts and/or finalization scripts has been tested – it can be inserted into ASAP database by using asap_db.pl file. This script accepts configuration file name as an input parameter and inserts everything from this configuration file into ASAP database. After that everything: plan script, query information, result scripts and/or finalization scripts can be accessed through database.

Example of configuration script:

```<QUESTION>
0
AA598665
```

```<SCRIPT>
# TEST script
# ID_script = 666
# <!--1--> ** Any EST accession number

query (TIGR, 1, $est_acc, 1)
result (TIGR, 1, $tc_id)

query (TIGR, 2, $tc_id, "Mouse")
result (TIGR, 2, $tc)

final(GO, 3, "output.txt", $est_acc, $tc)
```
<QUERY>
{TIGR, 1}
[parameters] gblist=<!--1--> species=<!--2--> Search=Search
[definition] Get tc ID for EST

{TIGR, 2}
[parameters] tc=<!--1--> species=<!--2-->
[definition] Get tc sequence for tc_ID

<Result>
{TIGR, 1}
GET(WHOLE, TABLE, cellpadding=5 & align=center & border=2, $TC=TRTD[1][5]);
if(!$get_result){
  ERROR("Error of parsing - maybe page format changed!");}

# Clear TC contig ID
GET($TC, A, 1, <!--1-->=TEXT);

{TIGR, 2}
GET(WHOLE, PRE, 1, <!--1-->=TEXT[1-]);

<FINAL>
{GO, 3}
PUT("<!--1-->", TEXT, 1, "<!--2-->\n\n")
2. Code

2.1. **User WEB Interface**

**Use cases:** Check Status, Ask Question

**Files:** main.pl, annotate.pl, status.pl, query.htm, status.htm

---

**main.pl – KERNEL OF USER WEB INTERFACE**

This is CGI script that is called when server can’t find html page that user is looking for. If the page has address ‘http://server/ASAP/query’ then script retrieves page based on template ‘query.htm’. If the page that user looks for has address ‘http://server/ASAP/status’ then script retrieves page based on template ‘status.htm’.

---

**annotate.pl – ASK QUESTION USE CASE**

**Actions:**

Page based on template ‘query.htm’ submits user information to the ‘annotate.pl’ CGI script. Script checks parameters inputted and if they are correct initiates search (put information about task to database and runs DISPATCHER) with this parameters, sends e-mail about successful initiation and returns page based on template ‘query.htm’ with inserted SUCCESS message. If inputted parameters are incorrect, script returns page based on template ‘query.htm’ with inserted FAILED message.

---

**Inputs:**

Name of input CGI parameters:

- **EST_LIST** – list of EST accession numbers or UniGene clusters
- **EST_FILE** – file with list of EST accession numbers or UniGene clusters
- **ORGANISM** – 0 (Hs) or 1 (Mm)
- **EMAIL** – e-mail of user
- **SA** – flag of simple annotation result (1-is needed, 0-isn’t needed)
- **GO1** – flag of function GO result (1-is needed, 0-isn’t needed)
- **GO2** – flag of process GO result (1-is needed, 0-isn’t needed)
- **GO3** – flag of component GO result (1-is needed, 0-isn’t needed)

---

**Functions:**

Function: 'ConnectDB'
Description: 'Connect to database'
Parameters: none
Return: 0 if error, 1 - otherwise

Function: 'DisconnectDB'
Description: 'Disconnect from database'
Parameters: none
Return: 1

Function: 'Make_sql'
Description: 'Function to run any SQL query to current $dbh (with prepare)'
Parameters:
$param -- string with SQL query
Return: 0 if error, 1 - otherwise

Function: 'E_mail'
Description: 'Function to send mail'
Parameters:
$to -- address of recipient
$subject -- subject of the letter
Return: 0 if success, 1 or 2 (depends on error) - otherwise
Outputs:
Page based on template ‘query.htm’. If inputted parameters correct and search initiated successfully, page has a message about success initiating. Message about error occurred otherwise.

**status.pl** – CHECK STATUS USE CASE

Actions:
Page based on template ‘status.htm’ submits user information to the ‘status.pl’ CGI script. Script checks parameter inputted and looks through database for task(s), defined by parameter. Script returns page based on template ‘status.htm’ with inserted SUCCESS message and information about status(es) of task(s) if this information found. Script returns page based on template ‘status.htm’ with inserted FAILED message otherwise.

Inputs:
Name of input CGI parameters:
EMAIL – task ID or e-mail of user to receive statuses for.

Functions:
- Function: 'ConnectDB'
  Description: 'Connect to database'
  Parameters: none
  Return: 0 if error, 1 - otherwise

- Function: 'DisconnectDB'
  Description: 'Disconnect from database'
  Parameters: none
  Return: 1

- Function: 'Make_sql'
  Description: 'Function to run any SQL query to current $dbh (with prepare)'
  Parameters:
  $string -- string with SQL query
  Return: 0 if error, 1 - otherwise

Outputs:
Page based on template ‘status.htm’. If status information found, page has a message about success initiating and status information. Message about error occurred otherwise.

2.2. **DISPATCHER**

Use cases: Receive answer
Files: huge_est.pl

**huge_est.pl** – INITIATES RECEIVE ANSWER USE CASE PROCESS

Actions:
Script receives parameters from ‘annotate.pl’ file, split list of accession numbers onto 1-entry chunks and initiates N searches at a time (runs MANAGER with initial parameters. N – is a number of searches allowed at a time). It waits until any of the searches is finished and runs next one. When every chunk is processed, script writes to database information about finishing task. After each chunk processed script writes to log file information about time of processing.
**Inputs:**
Command line parameters:
0 – name of input file with parameters of searching (created by ‘annotate.pl’)
1 – ten-digit identification number for user
2 – user mail
3 – plan (radio value from html or CUSTOM number) number.
4 – organism of searching
5 – reserved
6 – domain name. E.g ‘linux3.fccc.edu’ or ‘insilico.fccc.edu’ where ASAP is.

**Functions:**

Function: 'Main_sub'
Description: 'Entry point takes parameters from program parameter string'
Parameters: none
Return: none

Function: 'Log'
Description: 'Procedure just for write string to a file'
Parameters: $string_buf -- string to write
Return: 0 if error, 1 - otherwise

Function: 'PutConfigToFile'
Description: 'Procedure just for write string to a file'
Parameters: $file_name -- file name to write
$string_buf -- string buffer
Return: 0 if error, 1 - otherwise

Function: 'ReadTmpl'
Description: 'TEMPLATE CONFIG file reading procedure'
Parameters: $file_name -- file name to read
Return: empty string if error, file content - otherwise

Function: 'ReadEstFile'
Description: 'EST file reading procedure'
Parameters: $file_name -- file name to read
Return: empty array of error, array with file strings otherwise

Function: 'E_mail'
Description: 'Function to send mail'
Parameters: $to -- address of recipient
$subject -- subject of the letter
Return: 0 if success, 1 or 2 (depends on error) - otherwise

Function: 'ConnectDB'
Description: 'Connect to database'
Parameters: none
Return: 0 if error, 1 - otherwise

Function: 'DisconnectDB'
Description: 'Disconnect from database'
Parameters: none
Return: 1
Function: 'Make_sql'
Description: 'Function to run any SQL query to current $dbh (with prepare)'  
Parameters:
$string – string with SQL query 
Return: 0 if error, 1 – otherwise 

Function: 'Delete_similar'
Description: 'Function to delete from file duplicate strings'  
Parameters:
$file_name -- file name to delete similar strings from 
Return: 0 if error, 1 – otherwise 

Function: Config_by_plan'
Description: Function to compose manager config file by plan number'  
Parameters:
$plan_html_number -- plan number (radio value from html or CUSTOM)  
$accession_number -- accession number  
$entry_organism -- organism  
$report_file -- report file 
Return: buffer string with config 

Function: 'Usage'
Description: 'Usage procedure'  
Parameters: none 
Return: none 

Function: 'Exit_sub'
Description: 'Procedure for error or message'  
Parameters: none 
Return: 0 

Outputs: 
Writes to database about task finishing. Writes log files (huge_est_logs/*.txt) about process: 
*_huge_est_log.txt – full log about processing  
*_huge_est_shortlog.txt – last string of full log  
*_output.txt – standard output of ASAP system working  
*_ is a user ID passed as an input parameter to the script. 

2.3. **MANAGER**

Use cases: Receive Answer, Test Script and Add/Modify/Delete script  
Files: manager.pl, asapdb.pl  

manager.pl – MAIN SCRIPT OF RECEIVE ANSWER OR TEST SCRIPT USE CASE PROCESS  
Actions:  
Script receives parameters from command line and begins plan script processing according to input parameters.  

Inputs:  
0 – input file name with parameters.  
1 – flag of database using. (0 –use database, 1 – use only input file information)  

Functions:  
  
Function: 'Main_sub'
Description: 'Entry point'  
Parameters: none 
Return: none
Function: 'Fulfil_script'
Description: 'Recursive procedure for script processing'
Parameters:
$question_script_ref -- ref to array with script (part of whole script maybe) to fulfil
$offset -- string number of first string of this subscript
$block_name -- name of this block (for last using)
Return: 0 – error, 1 – all is ok, 2 – exit, other – exit from block 'other'

Function: 'Collect_block'
Description: 'Procedure is for get all stuff before } bracer'
Parameters:
$question_script_ref -- ref to array with script strings
$offset -- offset of block beginning
Return: (-1, undef) if error happened, (block length, block array) - otherwise

Function: 'Skip_block'
Description: 'Procedure is for skip all stuff before } bracer'
Parameters:
$question_script_ref -- ref to array with script strings
$offset -- offset of block beginning
Return: -1 if error happened, block length - otherwise

Function: 'GetSection'
Description: 'Procedure for getting section lines from file'
Parameters:
$i_file -- name of input file
$sect_name -- name of section to receive
Return: 0 – if error. Section string buffer or array of lines depends on context.

Function: 'GetQuestionParams'
Description: 'Procedure for cleaning question params from input file'
Parameters:
$sect_array_ref -- reference to array with params from file to clean
Return: 1

Function: 'GetQuestionScriptDB'
Description: 'Procedure for getting plan from database or input file and insert there some initial params'
Parameters:
$q_num -- QUESTION number
@q_par -- initial params
Return: 0 – if error, array of plan script strings otherwise

Function: 'Parse_internals'
Description: 'It parses structure like xxx,'zz,zz' into @arr = ((xxx), (zz,zz))'
Parameters:
$sign -- divider to split
internals -- income structure
oline_num -- number of line
$eval_prefix -- if present - package name to take variable values from to evaluate
Return: array of parsed structure

Function: 'ConnectDB'
Description: 'Connect to database'
Parameters: none
Return: 0 – if error, 1 – otherwise

Function: 'DisconnectDB'
Description: 'Disconnect from database'
Parameters: none
Return: 1
Function: 'Make_sql'
Description: 'Function to run any SQL query to current $dbh (with prepare)'
Parameters:
[string] -- string with SQL query
Return: 0 – if error, 1 - otherwise

Function: 'db_get'
Description: 'Function to get entries from DB (script function)'
Parameters:
$table -- table name to select from
$ID -- ID of the raw
$out -- reference to output array to put result in
Return: 0 – if error, number of entries found otherwise

Function: 'db_put'
Description: 'Function to put entries to DB (script function)'
Parameters:
$table -- table name to put to
$ID -- ID of the raw
$out -- input array
Return: 0 – if error, 1 - otherwise

Function: 'db_del'
Description: 'Function to del entries from DB (script function)'
Parameters:
$table -- table name to delete from
$ID -- ID of the raw to delete (delete all rows in case
Return: 0 – if error, 1 - otherwise

Function: 'Usage'
Description: 'Usage procedure'
Parameters: none
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
Exits with code 0 if error happened. Otherwise – 1 is exit code.

asapdb.pl – MAIN SCRIPT OF ‘ADD/MODIFY/DELETE SCRIPT’ USE CASE PROCESS
Actions:
Script receives parameters from command line and inserts plan script, query information, result scripts, finalization scripts into database. manager.pl uses some functions of this script to insert error and question information.

Inputs:
Nothing or:
0 – config file name to write information to database from.
1 – name of the file with ‘bitool’ information (can be omitted)
Functions:

Function: 'Main_sub'
Description: 'Entry point'
Parameters: none
Return: none

Function: 'Insert_error'
Description: 'Procedure to fill database (error)'
Parameters:
$ID_question -- ID of question
$err_package -- package where error occurred
$err_code -- code of error
$err_description -- description of error
$err_details -- error details
$err_date -- error date
Return: 0 if error, error ID otherwise

Function: 'Insert_question'
Description: 'Procedure to fill database (question, question_param)'
Parameters:
@question_buffer -- array of text strings from config file
Return: 0 if error, question ID otherwise

Function: 'Insert_script'
Description: 'Procedure to fill database (plan script, parameter, use_query, use_result, use_final)'
Parameters:
@script_buffer -- array of text strings from config file
Return: 0 if error, plan ID otherwise

Function: 'Insert_query'
Description: 'Procedure to fill database (query, query_of, const_param, mb use_query)'
Parameters:
@query_buffer -- array of text strings from config file
Return: 0 if error, query ID otherwise

Function: 'Insert_result'
Description: 'Procedure to fill database (result, mb use_result)'
Parameters:
@result_buffer -- array of text strings from config file
Return: 0 if error, result ID otherwise

Function: 'Insert_final'
Description: 'Procedure to fill database (final, mb use_final)'
Parameters:
@final_buffer -- array of text strings from config file
Return: 0 if error, finalization ID otherwise

Function: 'Insert_sites'
Description: 'Procedure to fill database (site, keyword, and has_keyword tables)'
Parameters:
$inp_file -- file name from which to take data
Return: none

Function: 'Make_sql'
Description: 'Function to run any SQL query to current $dbh (with prepare)'
$string -- string with SQL query
Return: 0 – if error, 1 – otherwise
Function: 'GetSection'
Description: 'Procedure for getting section lines from file'
Parameters:
$i_file -- name of input file
$sect_name -- name of section to receive
Return: 0 – if error. Section string buffer or array of lines depends on context.

Function: 'GetSubSection'
Description: 'Procedure for getting section lines from file'
Parameters:
$i_file -- name of input file
$sect_name -- name of section to receive
$id1 -- sub_section ID1
$id2 -- sub_section ID2
Return: 0 – if error. Section string buffer or array of lines depends on context.

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
Return: 0

Outputs:
Puts to database information

2.4. **CONFIG generator**
Use cases: Receive Answer, Test Script
Files: config_gener.pl

**config_gener.pl** – SCRIPT TO GENERATE CONFGS FOR APPROPRIATE GENERATORS

Actions:
‘manager.pl’ passes information to this script. Depends on what it is, config generator get from database appropriate information and puts it into configuration file for query, result or finalization generator and runs this generator. Collects all errors if happened.

Inputs:
none

Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'Main_sub'
Description: 'Program to create config for QUERY making, RESULTS processing, REPORT composing. Entry point'
Parameters:
$function_type -- qr - question-result pair, fin - final function
$first_params_ref -- ref to array with 1st function parameters
$second_params_ref -- ref to array with 2nd function parameters
$input_file -- name of config file (if non-DB using)
Return: 0 if error, 1 - otherwise

Function: 'Call_qr'
Description: 'QR function type'
Parameters:
$query_params_ref -- ref to array with query parameters
$result_params_ref -- ref to array with result parameters
$input_file -- name of config file (if non-DB using)

Return: 0 if error, 1 - otherwise

Function: 'Call_fin'
Description: 'FIN function type'
Parameters:

Function: 'GetSubSection'
Description: 'Procedure for getting section lines from file'
Parameters:

Function: 'PutConfigToFile'
Description: 'Procedure just for write string to a file'
Parameters:

Function: 'GetQueryConfigDB'
Description: 'Procedure for getting query config from database and insert there some necessary params'
Parameters:

Function: 'GetResultConfigDB'
Description: 'Procedure for getting result config from database and insert there some necessary params'
Parameters:

Function: 'GetFinalConfigDB'
Description: 'Procedure for getting query config from database and insert there some necessary params'
Parameters:

Function: 'Txt_to_html'
Description: 'Procedure converts any string to html format (e.g \n = %0d ...)' Parameters:

Function: 'ConnectDB'
Description: 'Connect to database'
Parameters: none

Return: 0 - if error, 1 - otherwise

Function: 'DisconnectDB'
Description: 'Disconnect from database'
Parameters: none
Return: 1

Function: 'Make_sql'
Description: 'Function to run any SQL query to current $dbh (with prepare)'
Parameters:
$string -- string with SQL query
Return: 0 - if error, 1 - otherwise

Function: 'Undef_package'
Description: 'Procedure for clearing all package variables'
Parameters:
$pack_name -- pack name to clear
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
Creates configuration files for generators.

2.5. QUERY module

Use cases: Receive Answer, Test Script, Receive Query, Give Result
Files: query_gener.pl, query_tmpl.pl, query_func.pl

query_gener.pl – SCRIPT TO GENERATE QUERY SCRIPT

Actions:
‘config_gener.pl’ passes config file name to this script. This script generates temporary file ‘*_query.pl’ based on template file ‘query_tmpl.pl’ (fills it with information from config file). Then returns to ‘config_gener.pl’ name of created file.

Inputs:
none

Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'Main_sub'
Description: 'QUERY script generator program. Entry point'
Parameters:
$input_file -- name of config file
$script_name_cfg -- name of result script file
$result_name_cfg -- name of result file after query
Return: 0 if error, created query file name - otherwise

Function: 'Read_config'
Description: 'Procedure for config reading (to global variables)'

Parameters:
$inp_file -- name of config file
Return: 0 if error, 1 - otherwise

Function: 'Usage'
Description: 'Usage procedure'
Parameters: none
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
Creates query script file. Returns name of this file.

query_tmpl.pl – TEMPLATE SCRIPT FOR QUERY SCRIPT GENERATING

Actions:
‘query_gener.pl’ loads this template and fills it with query parameters. New created script based on this template is run by ‘config_generator.pl’ and makes query to remote server, receives result from it and puts it into temporary file for result script. This script uses functions from ‘query_func.pl’ script.

Inputs:
none

Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'Main_sub'
Description: 'Entry point'
Parameters: none
Return: 0 if error, file name with result in it - otherwise

Function: 'Undef_package'
Description: 'Procedure for clearing all package variables'
Parameters:
$pack_name -- pack name to clear
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
Creates file with server response in it. Returns name of this file.
query_func.pl — SCRIPT WITH FUNCTIONS TO PROVIDE TO QUERY SCRIPT

Actions:
‘*_query.pl’ temporary script uses functions from this script.

Inputs:
none

Functions:

Function: 'Connect'
Description: 'Connection procedure (redirects if HTML or HTTP redirections)'
Parameters:
$host_name -- host name to connect
$get_line -- extension after host name
$method – query method: GET or POST
Return: 0 if error, string buffer with result - otherwise

Function: 'Tag_starts'
Description: 'Event triggered if <TAG PARAM=VALUE ...> is recognized. Check HTML redirections'
Parameters:
$tag -- TAG name
$attr_hash_ref -- HASH reference with parameters' name-value pairs
$self -- reference to $parser value
Return: none

Function: 'Tag_ends'
Description: 'Event triggered if </TAG> is recognized'
Parameters:
$tag -- TAG name
$self -- reference to $parser value
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
none

2.6. RESULT module

Use cases: Receive Answer, Test Script
Files: result_gener.pl, result_tmpl.pl, result_func.pl

result_gener.pl — SCRIPT TO GENERATE RESULT PROCESSING SCRIPT

Actions:
‘config Gener.pl’ passes config file name to this script. This script generates temporary file ‘*_result.pl’ based on template file ‘result_tmpl.pl’ (fills it with parsing information from config file). Then returns to ‘config_gener.pl’ name of created file.

Inputs:
none
Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'Main_sub'
Description: 'RESULT processing script generator program. Entry point'
Parameters:
$input_file -- name of config file
$script_name_cfg -- name of result script file
Return: 0 if error, created query file name - otherwise

Function: 'Read_config'
Description: 'Procedure for config reading (to global variables)'
Parameters:
$inp_file -- name of config file
Return: 0 if error, 1 - otherwise

Function: 'Convert_config'
Description: 'Procedure for converting result script config to perl code'
Parameters:
$cfg_buf_ref -- reference to array of config strings
$offset -- string number of first string of this subscript
$level -- to format with TABs easily
Return: 0 – if error, perl code - otherwise

Function: 'Parse_internals'
Description: 'It parses structure like xxx,'zz,zz' into @arr = ((xxx), (zz,zz))'
Parameters:
$sign -- divider to split
$internals -- income structure
$line_num -- number of line
Return: array of parsed structure

Function: 'Collect_block'
Description: 'Procedure for get all stuff before } bracer'
Parameters:
$question_script_ref -- ref to array with script strings
$offset -- offset of block beginning
Return: (-1, undef) if error happened, (block length, block array) - otherwise

Function: 'Usage'
Description: 'Usage procedure'
Parameters: none
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
Creates result processing script file. Returns name of this file.
**result_tmpl.pl** - TEMPLATE SCRIPT FOR RESULT PROCESSING SCRIPT GENERATING

**Actions:**
'`result_gener.pl`' loads this template and fills it with query parameters. New created script based on this template is run by `config_generator.pl` and parses result received by `*_query.pl` from temporary file. This script uses functions from `result_func.pl` script.

**Inputs:**
none

**Functions:**

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'Main_sub'
Description: 'Entry point'
Parameters:
$input_file -- name of file to process
$result_params_ref -- reference to array with var name to evaluate
Return: 0 if error, file name with result in it - otherwise

Function: 'Undef_package'
Description: 'Procedure for clearing all package variables'
Parameters:
$pack_name -- pack name to clear
Return: none

Function: 'Step_eval'
Description: 'Procedure for step evaluation (evaluates all variables that were in GET output)'
Parameters:
$get_type -- type of GET function (0 - just GET, 1 - GET_ALL)
$entry_number -- number of variables to evaluate
2 -- name of first variable to evaluate
3 -- name of second variable to evaluate
4 -- . . .
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

**Outputs:**
Evaluates variables for next step of plan script processing

**result_func.pl** - SCRIPT WITH FUNCTIONS TO PROVIDE TO RESULT PROCESSING SCRIPT

**Actions:**
`*_result.pl` temporary script uses functions from this script.
Inputs:
none

Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'GET'
Description: 'Procedure for GET method. All GET functionality'
Parameters:
$res_buff -- reference to chunk of HTML text to process
$tag_nam -- FRAME, TABLE, A, TEXT
$cond -- NAME="xxx" or TEXT="xxx" or NUMBER=15 (& and | allowed)
$outp -- NAME=$xxx or TEXT=$xxx or NUMBER=$xxx (& allowed)
$kind -- 0 - first found chunk, 1 - all suitable cases
Return: 0 if error, number of entries found - otherwise

Function: 'Basic_checkout'
Description: 'Find chunk, check condition, format output. Based on HTML::Parser perl module. See documentation on it'
Parameters:
$tag -- TAG name
$attr_hash_ref -- HASH reference with parameters' name-value pairs
$offset -- offset from the beginning of the buffer (after last <>)
Return: 0 if error, 1 - otherwise

Function: 'Tag_starts'
Description: 'Event triggered if <TAG PARAM=VALUE ...> is recognized'
Parameters:
$tag -- TAG name
$attr_hash_ref -- HASH reference with parameters' name-value pairs
$self -- reference to $parser value
$offset -- offset from the beginning of the buffer (after last <>)
$text -- text (not decrypted)
$dtext -- text (decrypted)
Return: none

Function: 'Tag_ends'
Description: 'Event triggered if </TAG> is recognized'
Parameters:
$tag -- TAG name
$text -- text (not decrypted)
$dtext -- text (decrypted)
Return: none

Function: 'Text_appears'
Description: 'Event triggered if simple text is recognized'
Parameters:
$text -- just TEXT - already decrypted ( &gt; means >)
$dtext -- text (decrypted)
Return: none

Function: 'End_document'
Description: 'Event triggered if end of chunk has reached'
Parameters:
$text -- just TEXT - with decrypted (or not :) symbols < > & "
$dtext -- text (decrypted)
Return: none
Function: 'Comment_appears'
Description: 'Event triggered if end of chunk has reached'
Parameters:
$text -- just TEXT - with decrypted (or not :) symbols < > & "
$dtext -- text (decrypted)
Return: none

Function: 'TBL_tag_starts'
Description: 'FOR TABLE TRTD: Event triggered if <TAG ...> is recognized'
Parameters:
$tag -- TAG name
$attr_hash_ref -- HASH reference with parameters' name-value pairs
$self -- reference to $tbl_parser value
$text -- original HTML text to collect in $nested_info
Return: none

Function: 'TBL_tag_ends'
Description: 'FOR TABLE TRTD: Event triggered if </TAG> is recognized'
Parameters:
$tag -- TAG name
$text -- original HTML text to collect in $nested_info
$self -- reference to $tbl_parser value
Return: none

Function: 'TBL_text_appears'
Description: 'FOR TABLE TRTD: Event triggered if simple text is recognized'
Parameters:
$text -- just TEXT - with crypted symbols < > & "
Return: none

Function: 'TBL_comment_appears'
Description: 'FOR TABLE TRTD: Event triggered if comment is recognized'
Parameters:
$text -- just TEXT - with crypted symbols < > & "
Return: none

Function: 'TBL_end_document'
Description: 'FOR TABLE TRTD: Event triggered if end of chunk has reached'
Parameters:
$text -- just TEXT - with crypted symbols < > & "
Return: none

Function: 'TBL_get_element'
Description: 'FOR TABLE TRTD: Event triggered if simple text is recognized'
Parameters: none
Return: none

Function: 'TBL_need_element'
Description: 'FOR TABLE TRTD: see whether we need this element or not'
Parameters:
$row -- if this row allowed
$col -- if this col allowed
Return: 0 – don’t need this element, 1 – otherwise

Function: 'TXT_tag_starts'
Description: 'FOR TEXT: Event triggered if <TAG ...> is recognized'
Parameters:
$tag -- TAG name
$attr_hash_ref -- HASH reference with parameters' name-value pairs
$self -- reference to $tbl_parser value
$text -- original HTML text to collect in $nested_info
Return: none
Function: 'TXT_tag_ends'
Description: 'FOR TEXT: Event triggered if </TAG> is recognized'
Parameters:
$tag -- TAG name
$text -- original HTML text to collect in $nested_info
$self -- reference to $tbl_parser value
Return: none

Function: 'TXT_textAppears'
Description: 'FOR TEXT: Event triggered if simple text is recognized'
Parameters:
$text -- just TEXT - with decrypted (or not :) symbols < > & 
Return: none

Function: 'TXT_commentAppears'
Description: 'FOR TEXT: Event triggered if comment is recognized'
Parameters:
$text -- just TEXT - with decrypted (or not :) symbols < > & 
Return: none

Function: 'TXT_end_html'
Description: 'FOR TEXT: Event triggered if if end of chunk has reached'
Parameters:
$text -- just TEXT - with decrypted (or not :) symbols < > & 
Return: none

Function: 'Parse_comma_dash'
Description: 'Procedure for parsing structure like x1,x2'
Parameters:
$cd_str -- comma, dash format string
Return: Two-dimensional array (2 rows). 1st row – 'from' values, 2nd row – 'to' values

Function: 'TXT_need_string'
Description: 'FOR TXT: see whether we need this string or not'
Parameters:
$str -- if this str allowed
Return: 0 – don’t need, 1 – otherwise.

Function: 'TXT_parse_common'
Description: 'FOR TXT: Get all text or html into $nested_info variable'
Parameters:
$tg_nd -- include tags?
$off_buf_ref -- reference to off_buffer
Return: none

Function: 'Is_match'
Description: 'Procedure for question answer whether two scalars matches'
Parameters:
$original -- first param for matching question
$pattern -- second param for matching question
Return: 2-element array. 1st element: 1-matches, 0-otherwise. 2nd element: if capturing is needed - array of captured atoms.

Function: 'Is_interrupt'
Description: 'Procedure checking if one tag interrupt another'
Parameters:
$tag -- tag w/o <>
$contender -- tag w/o <> and w / before ( if it closing tag need )
Return: 0 – doesn’t interrupt, 1 – interrupts.

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
none

2.7. **FINALIZATION module**

Use cases: Receive Answer, Test Script

Files: final_gener.pl, final_tmpl.pl, final_func.pl

**finalGener.pl** – SCRIPT TO GENERATE REPORT COMPOSING SCRIPT

Actions:
'config_gener.pl' passes config file name to this script. This script generates temporary file `*_final.pl` based on template file `final_tmpl.pl` (fills it with parsing information from config file). Then returns to `config_gener.pl` name of created file.

Inputs:
none

Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'Main_sub'
Description: 'FINALIZATION script generator program. Entry point'
Parameters:
$input_file -- name of config file
$script_name_cfg -- name of result script file
Return: 0 if error, created query file name - otherwise

Function: 'Read_config'
Description: 'Procedure for config reading (to global variables)'
Parameters: $inp_file -- name of config file
Return: 0 if error, 1 - otherwise

Function: 'Convert_config'
Description: 'Procedure for converting result script config to perl code'
Parameters: $cfg_buf_ref -- reference to array of config strings
$offset -- string number of first string of this subscript
$level -- to format with TABs easily
Return: 0 - if error, perl code - otherwise

Function: 'Parse_internals'
Description: 'It parses structure like xxx,'zz,zz' into @arr = ((xxx), (zz,zz))'
Parameters: $sign -- divider to split
$internals -- income structure
$line_num -- number of line
Return: array of parsed structure
Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
Creates finalization script file. Returns name of this file.

final_tmpl.pl – TEMPLATE SCRIPT FOR FINALIZATION SCRIPT GENERATING
Actions:
‘final_gener.pl’ loads this template and fills it with query parameters. New created script based on this template is run by ‘config_generator.pl’ and composes report. This script uses functions from ‘final_func.pl’ script.

Inputs:
none

Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'Main_sub'
Description: 'Entry point'
Parameters:
$result_params_ref -- reference to array with var name to evaluate
Return: 0 if error, 1 - otherwise

Function: 'Undef_package'
Description: 'Procedure for clearing all package variables'
Parameters:
$pack_name -- pack name to clear
Return: none

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
Puts string(s) of composed report in a file.

final_func.pl – SCRIPT WITH FUNCTIONS TO PROVIDE TO FINALIZATION SCRIPT
Actions:
‘*_final.pl’ temporary script uses functions from this script.

Inputs:
none
Functions:

Function: 'Def_init'
Description: 'Initialize some variables'
Parameters: none
Return: none

Function: 'PUT'
Description: 'Procedure for creating file (appending to existing) and writing there composed report'
Parameters:
$file_name -- file name to put result into
$type_name -- type of input variable formatting
$attributes -- some attributes: for every type their differ
$input_var -- values to put
Return: 0 if error, 1 - otherwise

Function: 'Exit_sub'
Description: 'Procedure for error or message'
Parameters:
$err_str -- string to put in error
$err_src -- source of error (1-this package, 0-another)
$err_code -- error code (see documentation)
Return: 0

Outputs:
none
3. Specials

3.1. Error codes

To identify error happened somewhere during ASAP system working next notation was accepted:
Error information contains error code.

Error code – 4-digit code. Digits are:

1st – number of module where error fires
  0 – blank
  1 – MANAGER
  2 – CONFIG GENERATOR
  3 – QUERY module
  4 – RESULT module
  5 – FINALIZATION module

2nd – type of module:
  0 – blank
  1 – generator (*_gener.pl files)
  2 – template (*_tmpl.pl files)
  3 – function (*_func.pl files)

3rd, 4th – error type:
  00 – not a error
  01 – IO error (file system)
  02 – script/config syntax/semantic errors
  03 – connection error (query module)
  04 – parsing error (result module)
  05 – unrecognized error
  06 – database error

So - codes of program packages:
  10 – man_man (manager.pl)
  20 – con_gen (gonfig_gener.pl)
  31 – que_gen (query_gener.pl)
  32 – que_fin (query_tmpl.pl)
  33 – que_fin_fun (query_func.pl)
  41 – res_gen (result_gener.pl)
  42 – res_fin (result_tmpl.pl)
  43 – res_fin_fun (result_func.pl)
  51 – fin_gen (final_gener.pl)
  52 – fin_fin (final_tmpl.pl)
  53 – fin_fin_fun (final_func.pl)
4. Requirements

Installation of ASAP system requires:

perl 5 or higher:
  additional libraries:
    DBI
    DBD::Mysql
    HTML::Parser

MySQL

HTTP server

Files and folders that should be in directory cgi-bin/ASAP on HTTP server:

Folders:
  huge_est_configs
  huge_est_finals
  huge_est_inputs
  huge_est_logs
  results

Files:
  main.pl
  annotate.pl
  status.pl
  query.htm
  status.htm
  huge_est.pl
  manager.pl
  config_gener.pl
  query_gener.pl
  query_tmpl.pl
  query_func.pl
  result_gener.pl
  result_tmpl.pl
  result_func.pl
  final_gener.pl
  final_tmpl.pl
  final_func.pl
Files and folders that should be in directory html/ASAP on HTTP server:

Folders:
reports

Files:
.htaccess

Content of `.htaccess` file:

Options -Indexes
ErrorDocument 404 /cgi-bin/ASAP/main.pl
ErrorDocument 403 /cgi-bin/ASAP/main.pl
ErrorDocument 401 /cgi-bin/ASAP/main.pl

MySQL should contain database structure from `asap.sql` file.