Test Plan

**Introduction**

This test approach document describes the appropriate strategies, process, workflows and methodologies used to plan, organize, execute and manage testing of software projects within caBIG.

## Scope

Describe the current test approach scope based on your role and project objectives.

### In Scope

The caBIG <workspace name> <system name> *Test Plan* defines the unit, integration, system, regression, and Client Acceptance testing approach. The test scope includes the following:

* Testing of all functional, application performance, security and use cases requirements listed in the *Use Case* document.
* Quality requirements and fit metrics<system name>.
* End-to-end testing and testing of interfaces of all systems that interact with the <system name>.

### Out of Scope

The following are considered out of scope for caBIG <workspace name> <system name> systemTest Plan and testing scope:

* Functional requirements testing for systems outside <application name>
* Testing of Business SOPs, disaster recovery and Business Continuity Plan.

## Quality Objective

### Primary Objective

A primary objective of testing application systems is to: ***assure that the system meets the full requirements, including quality requirements (AKA: Non-functional requirements) and fit metrics for each quality requirement and satisfies the use case scenarios and maintain the quality of the product.*** At the end of the project development cycle, the user should find that the project has met or exceeded all of their expectations as detailed in the requirements.

Any changes, additions, or deletions to the requirements document, Functional Specification, or Design Specification will be documented and tested at the highest level of quality allowed within the remaining time of the project and within the ability of the test team.

### Secondary Objective

The secondary objective of testing application systems will be to: ***identify and expose all issues and associated risks, communicate all known issues to the project team, and ensure that all issues are addressed in an appropriate matter before release.*** As an objective, this requires careful and methodical testing of the application to first ensure all areas of the system are scrutinized and, consequently, all issues (bugs) found are dealt with appropriately.

## Roles and Responsibilities

Roles and responsibilities may differ based on the actual SOW. Below listed functions are for testing phase.

### Developer

An NCI-designated Cancer Center selected and funded by NCICB to participate in a specific Workspace to undertake software or solution development activities. Responsible to:

(a) Develop the system/application

(b) Develop Use cases and requirements in collaboration with the Adopters

(c) Conduct Unit, system, regression and integration testing

(d) Support user acceptance testing

### Adopter

An NCI-designated Cancer Center selected and funded by NCICB to undertake formal adoption, testing, validation, and application of products or solutions developed by Workspace Developers. Responsible to:

(a) Contribute to Use case, requirement development through review

(b) Contribute to develop and execution of the development test scripts through review

(c) Conduct Full User Acceptance, regression, and end-to-end testing; this includes identifying testing scenarios, building the test scripts, executing scripts and reporting test results

### Testing Process Management Team

Include NCI, BAH and Cancer Center Leads allocated to the <workspace name>. Group responsible to manage the entire testing process, workflow and quality management with activities and responsibilities to:

(a) Monitor and manage testing integrity and Support testing activities

(b) Coordinate activities across cancer centers

Add more as appropriate to testing scope

## Assumptions for Test Execution

Below are some minimum assumptions (in black) that has be completed with some examples (in red). Any example may be used if deemed appropriate for the particular project. New assumptions may also be added that are reasoned to be suitable to the project.

* For User Acceptance testing, the Developer team has completed unit, system and integration testing and met all the Requirement’s (including quality requirements) based on Requirement Traceability Matrix.
* User Acceptance testing will be conducted by End-users
* Test results will be reported on daily basis using Gforge. Failed scripts and defect list from Gforge with evidence will be sent to Developer directly.
* Use cases have been developed by Adopters for User Acceptance testing. Use cases are approved by test lead.
* Test scripts are developed and approved.
* Test Team will support and provide appropriate guidance to Adopters and Developers to conduct testing
* Major dependencies should be reported immediately after the testing kickoff meeting.

## Constraints for Test Execution

Below are some minimum assumptions (in black) followed by example constraints (red). Any example may be used if deemed appropriate for the particular project. New constraints may also be added that are reasoned to be suitable to the project.

* Adopters should clearly understand on test procedures and recording a defect or enhancement. Testing Process Management Team will schedule a teleconference with Developers and Adopters to train and address any testing related issues.
* Developer will receive consolidated list of request for test environment set up, user accounts set up, data set (actual and mock data), defect list, etc. through **GForge** after the initial Adopter testing kick off meeting.
* Developer will support ongoing testing activities based on priorities
* Test scripts must be approved by Test Lead prior test execution
* Test scripts, test environment and dependencies should be addressed during testing kickoff meeting in presence of a SME and request list should be submitted within 3 days of the kickoff meeting
* The Developer cannot execute the User Acceptance and End to End test scripts. After debugging, the developer can conduct their internal test, but no results from that test can be recorded / reported.
* Adopters are responsible to identify dependencies between test scripts and submit clear request to set up test environment

## Definitions

Bugs: Any error or defect that cause the software/application or hardware to malfunction. That is also included in the requirements and does not meet the required workflow, process or function point.

Enhancement:

1) Any alteration or modification to the existing system for better workflow and process.

2) An error or defect that causes the software/application or hardware to malfunction.

Where 1) and 2) is NOT included in the requirements can be categorized as an enhancement.

Enhancement can be added as a new requirement after appropriate Change Management process.

# Test Methodology

## Purpose

### Overview

The below list is not intended to limit the extent of the test plan and can be modified to become suitable for the particular project.

The purpose of the Test Plan is to achieve the following:

1. Define testing strategies for each area and sub-area to include all the functional and quality (non-functional) requirements.
2. Divide Design Spec into testable areas and sub-areas (do not confuse with more detailed test spec). Be sure to also identify and include areas that are to be omitted (not tested) also.
3. Define bug-tracking procedures.
4. Identify testing risks.
5. Identify required resources and related information.
6. Provide testing Schedule.

### Usability Testing

The purpose of usability testing is to ensure that the new components and features will function in a manner that is acceptable to the customer.

Development will typically create a non-functioning prototype of the UI components to evaluate the proposed design. Usability testing can be coordinated by testing, but actual testing must be performed by non-testers (**as close to end-users as possible).** Testing will review the findings and provide the project team with its evaluation of the impact these changes will have on the testing process and to the project as a whole.

### Unit Testing (Multiple)

Unit Testing is conducted by the Developer during code development process to ensure that proper functionality and code coverage have been achieved by each developer both during coding and in preparation for acceptance into iterations testing.

The following are the example areas of the project must be unit-tested and signed-off before being passed on to regression Testing:

1. Databases, Stored Procedures, Triggers, Tables, and Indexes
2. NT Services
3. Database conversion
4. .OCX, .DLL, .EXE and other binary formatted executables

### Iteration/Regression Testing

During the repeated cycles of identifying bugs and taking receipt of new builds (containing bug fix code changes), there are several processes which are common to this phase across all projects. These include the various types of tests: functionality, performance, stress, configuration, etc. There is also the process of communicating results from testing and ensuring that new drops/iterations contain stable fixes (regression). The project should plan for a minimum of 2-3 cycles of testing (drops/iterations of new builds).

At each iteration, a debriefing should be held. Specifically, the report must show that to the best degree achievable during the iteration testing phase, all identified severity 1 and severity 2 bugs have been communicated and addressed. At a minimum, all priority 1 and priority 2 bugs should be resolved prior to entering the beta phase.

Below are examples. Any example may be used if deemed appropriate for the particular project. New content may also be added that are reasoned to be suitable to the project.

Important deliverables required for acceptance into Final Release testing include:

1. Application SETUP.EXE
2. Installation instructions
3. All documentation (beta test scripts, manuals or training guides, etc.)

### Final release Testing

Testing team with end-users participates in this milestone process as well by providing confirmation feedback on new issues uncovered, and input based on identical or similar issues detected earlier. The intention is to verify that the product is ready for distribution, acceptable to the customer and iron out potential operational issues.

Assuming critical bugs are resolved during previous iterations testing- Throughout the Final Release test cycle, bug fixes will be focused on minor and trivial bugs (severity 3 and 4). Testing will continue its process of verifying the stability of the application through regression testing (existing known bugs, as well as existing test cases).

The milestone target of this phase is to establish that the application under test has reached a level of stability, appropriate for its usage (number users, etc.), that it can be released to the end users and caBIG community.

### Testing completeness Criteria

Release for production can occur only after the successful completion of the application under test throughout all of the phases and milestones previously discussed above.

The milestone target is to place the release/app (build) into production after it has been shown that the app has reached a level of stability that meets or exceeds the client expectations as defined in the Requirements, Functional Spec., and caBIG Production Standards.

## Test Levels

Testing of an application can be broken down into three primary categories and several sub-levels. The three primary categories include tests conducted every build (Build Tests), tests conducted every major milestone (Milestone Tests), and tests conducted at least once every project release cycle (Release Tests). The test categories and test levels are defined below:

### Build Tests

#### Level 1 - Build Acceptance Tests

Build Acceptance Tests should take less than 2-3 hours to complete (15 minutes is typical). These test cases simply ensure that the application can be built and installed successfully. Other related test cases ensure that adopters received the proper Development Release Document plus other build related information (drop point, etc.). The objective is to determine if further testing is possible. If any Level 1 test case fails, the build is returned to developers un-tested.

#### Level 2 - Smoke Tests

Smoke Tests should be automated and take less than 2-3 hours (20 minutes is typical). These tests cases verify the major functionality a high level.

The objective is to determine if further testing is possible. These test cases should emphasize breadth more than depth. All components should be touched, and every major feature should be tested briefly by the Smoke Test. If any Level 2 test case fails, the build is returned to developers un-tested.

#### Level 2a - Bug Regression Testing

Every bug that was “Open” during the previous build, but marked as “Fixed, Needs Re-Testing” for the current build under test, will need to be regressed, or re-tested. Once the smoke test is completed, all resolved bugs need to be regressed. It should take between 5 minutes to 1 hour to regress most bugs.

### Milestone Tests

#### Level 3 - Critical Path Tests

Critical Path test cases are targeted on features and functionality that the user will see and use every day.

Critical Path test cases must pass by the end of every 2-3 Build Test Cycles. They do not need to be tested every drop, but must be tested at least once per milestone. Thus, the Critical Path test cases must all be executed at least once during the Iteration cycle, and once during the Final Release cycle.

### Release Tests

#### Level 4 - Standard Tests

Test Cases that need to be run at least once during the entire test cycle for this release. These cases are run once, not repeated as are the test cases in previous levels. Functional Testing and Detailed Design Testing (Functional Spec and Design Spec Test Cases, respectively). These can be tested multiple times for each Milestone Test Cycle (Iteration, Final Release, etc.).

Standard test cases usually include Installation, Data, GUI, and other test areas.

#### Level 5 - Suggested Test

These are Test Cases that would be nice to execute, but may be omitted due to time constraints.

Most Performance and Stress Test Cases are classic examples of Suggested test cases (although some should be considered standard test cases). Other examples of suggested test cases include WAN, LAN, Network, and Load testing.

## Bug Regression

Bug Regression will be a central tenant throughout all testing phases.

All bugs that are resolved as “Fixed, Needs Re-Testing” will be regressed when Testing team is notified of the new drop containing the fixes. When a bug passes regression it will be considered “Closed, Fixed”. If a bug fails regression, adopters testing team will notify development team by entering notes into GForge. **When a Severity 1 bug fails regression, adopters Testing team should also put out an immediate email to development.** The Test Lead will be responsible for tracking and reporting to development and product management the status of regression testing.

## Bug Triage

Bug Triages will be held throughout all phases of the development cycle. Bug triages will be the responsibility of the Test Lead. Triages will be held on a regular basis with the time frame being determined by the bug find rate and project schedules.

Thus, it would be typical to hold few triages during the **Planning phase**, then maybe one triage per week during the **Design phase**, ramping up to twice per week during the latter **stages of the Development phase**. Then, the Stabilization phase should see a substantial reduction in the number of new bugs found, thus a few triages per week would be the maximum (to deal with status on existing bugs).

The Test Lead, Product Manager, and Development Lead should all be involved in these triage meetings. The Test Lead will provide required documentation and reports on bugs for all attendees. The purpose of the triage is to determine the type of resolution for each bug and to prioritize and determine a schedule for all “To Be Fixed Bugs’. Development will then assign the bugs to the appropriate person for fixing and report the resolution of each bug back into the GForge bug tracker system. The Test Lead will be responsible for tracking and reporting on the status of all bug resolutions.

## Suspension Criteria and Resumption Requirements

This section should be defined to list criteria’s and resumption requirements should certain degree and pre-defined levels of test objectives and goals are not met.

Please see example below:

- Testing will be suspended on the affected software module when Smoke Test (Level 1) or Critical Path (Level 2) test case bugs are discovered after the 3rd iteration.

- Testing will be suspended if there is critical scope change that impacts the Critical Path

A bug report should be filed by Development team. After fixing the bug, Development team will follow the drop criteria (described above) to provide its latest drop for additional Testing. At that time, adopters will regress the bug, and if passes, continue testing the module.

## Test Completeness

Testing will be considered complete when the following conditions have been met:

### Standard Conditions:

1. When Adopters and Developers, agree that testing is complete, the app is stable, and agree that the application meets functional requirements.
2. Script execution of all test cases in all areas have passed.
3. Automated test cases have in all areas have passed.
4. All priority 1 and 2 bugs have been resolved and closed.
5. NCI approves the test completion
6. Each test area has been signed off as completed by the Test Lead.
7. 50% of all resolved severity 1 and 2 bugs have been successfully re-regressed as final validation.
8. Ad hoc testing in all areas has been completed.

### Bug Reporting & Triage Conditions:

Please add Bug reporting and triage conditions that will be submitted and evaluated to measure the current status.

1. Bug find rate indicates a decreasing trend prior to Zero Bug Rate (no new Sev. 1/2/3 bugs found).
2. Bug find rate remains at 0 new bugs found (Severity 1/2/3) despite a constant test effort across 3 or more days.
3. Bug severity distribution has changed to a steady decrease in Sev. 1 and 2 bugs discovered.
4. No ‘Must Fix’ bugs remaining prior despite sustained testing.

**REVISIONS HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision No.** | **Date of Issue** | **Author** | **Description** |
| 1. | 19.08.2014 | <Tester Name> | Functional testing |
| 2. | 03.09.2014 | <Tester Name> | Updated pre steps |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Browser  Build  version | Google Chrome | Mozilla Firefox | Internet Explorer 8 | Internet Explorer 11 |
|  |  |  |  |  |
|  |  |  |  |  |

# FUNCTIONAL TEST CASES

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Steps** | **Expected results** | **Test results** | | | |
|  | **Open component.** |  | **Chrome** | **Firefox** | **IE 8** | **IE 11** |
|  | **Pre-steps:**   1. Run Ajax File Browser from Visual Studio. |  |  |  |  |  |
| **1.** | **Uploading by clicking on the button on toolbar.** |  |  |  |  |  |
| 1.1. | 1. Click on “Upload” button on toolbar. 2. Select a file “Open” dialog and push “Open” button. | Selected file is uploaded and has same size as original local file. |  |  |  |  |
| 1.2. | 1. Click on “Upload Folder” on toolbar. 2. Select a folder on “Browse folder” dialog and push “OK” button. | Selected folder is uploaded and has same size as original local folder. |  |  |  |  |
| **2.** | **Uploading by using a context menu.** |  |  |  |  |  |
| 2.1. | 1. Right click on any folder on the list of files and folders. 2. Select “Upload Files” from opened context menu. 3. Select a file on “Open” dialog and push “Open” button. | Selected file is uploaded and has same size as original local file. |  |  |  |  |
| 2.2. | 1. Right click on any folder on the list of files and folders. 2. Select “Upload Folder” from opened context menu. 3. Select a folder on “Browse folder” dialog and push “OK” button. | Selected folder is uploaded and has same size as original local folder. |  |  |  |  |
| 2.3. | 1. Upload “Test.doc” file on AjaxBrowser using “Upload” button on toolbar. 2. Open locally “Test.doc” and make some changes to it, and save this changes. 3. Right click on “Test.doc” file on list of files AjaxBrowser. 4. Select “Update File” from context menu. 5. And choose changed local copy of file. 6. Push “Open” on “Open” dialog 7. Click “Yes” on “Overwrite Confirmation” dialog. 8. Open changed “Test.doc” in Ajax File Browser web page. | “Test.doc” file is updated and has same size as original local file. |  |  |  |  |
| **3.** | **Uploading using “Browse” button on “Upload Progress Panel”.** |  |  |  |  |  |
| 3.1. | 1. Click on “Browse” button on “Upload Progress Panel”. 2. Select a file on “Open” dialog and push “Open” button. | Selected file is uploaded and has same size as original local file. |  |  |  |  |
| **4.** | **Uploading by using drag & drop.** |  |  |  |  |  |
| 4.1. | 1. Select a file on your file system. 2. Drop selected file to list of files and folders of Ajax File Browser on web browser. | Selected file is uploaded and has same size as original local file. | Not ask for rename or owerwrite exist file |  |  |  |
| 4.2. | 1. Select a folder on your file system. 2. Drop selected folder to list of files and folders of Ajax File Browser on web browser. | Selected folder is uploaded and has same size as original local folder. | Selected folder is uploaded  and has same size as original  local folder |  |  |  |
| **5.** | **Uploading if file/folder already exist.** |  |  |  |  |  |
| 5.1. | 1. Upload “Test.doc” file on “Home” page of Ajax File Browser. 2. Create another “Test.doc” file and try to upload it using ”Upload” button on toolbar. 3. Click “Yes” on “Overwrite Confirm” dialog. 4. Open uploaded file from the list on “Ajax Browser”. | New copy of file is uploaded. Compare size of local file with new copy that has been uploaded-they should be the same. |  |  |  |  |
| 5.2. | 1. Upload “Test.doc” file on “Home” page of Ajax File Browser using “Upload” button on toolbar. 2. Create another “Test.doc” file and try to upload it using. 3. ”Upload” button on toolbar. Click “No” on “Overwrite Confirm” dialog. 4. Open “Test.doc” from list of files on “Ajax Browser”. | “Test.doc” file isn’t changed. Compare size of local "Test.doc" file with file on the list of AjaxBrowser - they should be the same. |  |  |  |  |
| 5.3. | 1. Upload “Test.doc” file on “Home” page of Ajax File Browser using “Upload” button on toolbar. 2. Create another “Test.doc” file and try to upload it using ”Upload” button on toolbar. 3. Click “Cancel” on “Overwrite Confirm” dialog. 4. Open “Test.doc” from list of files on “Ajax Browser”. | “Test.doc” file isn’t changed. Compare size of local "Test.doc" file with file on the list of AjaxBrowser - they should be the same. |  |  |  |  |
| 5.4. | 1. Upload or create “Test” folder on list of files and folders using “Upload folder” button on toolbar. Create “Test” folder on your local drive. 2. Click on “Upload Folder” on toolbar of “Ajax Browser” page. 3. Select previously created locally “Test” folder and click “Ok” button on “Browse folder” dialog. 4. Click “Yes” on “Overwrite Confirm” dialog. 5. Look at “Test” folder on list of files and folders. | “Test” folder is overwritten by new copy. To verify it look at “Date Modified” of that folder and verify that it is changed after overwriting. |  |  |  |  |
| 5.5. | 1. Upload several files using "Upload" button on toolbar. 2. Select same files that just has been uploaded locally on your file system. Drop selected files to list of files on Ajax File Browser. 3. Click "Yes to All" on "Overwrite Confirm" dialog. 4. Check all files that should be overwritten. | All files should be changed. Check "Date Modified" of every file and verify that time of modifying on all files is changed. |  |  |  |  |
| 5.6. | 1. Upload several files using "Upload" button on toolbar. 2. Select same files that just has been uploaded locally on your file system. 3. Drop selected files to list of files on Ajax File Browser. 4. Click "Yes" on every "Overwrite Confirm" dialog. | "Overwrite Confirm" dialog appears for every file overwriting. All files should be changed. Check "Date Modified" of every file and verify that time of modifying on all files is changed. |  |  |  |  |
| 5.7. | 1. Upload several files using "Upload" button on toolbar. 2. Select same files that just has been uploaded locally on your file system. 3. Drop selected files to list of files on Ajax File Browser. 4. Click "No to All" on "Overwrite Confirm" dialog. 5. Check all files that should be overwritten. | All files shouldn't be changed. Check "Date Modified" of every file and verify that time of modifying on all files isn't changed. |  |  |  |  |
| **6.** | **Uploading of few files/folders.** |  |  |  |  |  |
| 6.1. | 1. Click on “Upload” button on toolbar. 2. Select few files (by holding “Ctrl” button) on “Open” dialog and push “Open” button. | All selected files are uploaded. Compare sizes of uploaded files with original local copies and verify that all local files are the same size as uploaded copies. |  |  |  |  |
| 6.2. | 1. Select few folders on your file system. 2. Drop selected folders to list of files and folders of Ajax File Browser on web browser. | All selected folders are uploaded. Compare sizes of uploaded files with original local copies and verify that all local files are the same size as uploaded copies. |  |  |  |  |
| **7.** | **Uploading to folder.** |  |  |  |  |  |
| 7.1. | 1. Create a folder “Test” on list of files using “New folder” button on toolbar. 2. Open “Test” folder. 3. Click on “Upload” button on toolbar. 4. Select a file or create new one on “Open” dialog and push “Open” button. | Selected file is uploaded to opened folder and size of this file is same as size of original local copy. |  |  |  |  |
| 7.2. | 1. Create a folder “Test” on list of files using “New folder” button on toolbar. 2. Open “Test” folder. 3. Create locally folder “Test1” and copy on it some files. 4. Click on “Upload folder” button on toolbar. 5. Select a folder “Test1” on “Browse folder” dialog and push “Ok” button. | Selected folder is uploaded to opened folder. Open uploaded folder and verify that all copied locally to folder files are present on uploaded folder. |  |  |  |  |
| 7.3. | 1. Create a folder “Test” on list of files using “New folder” button on toolbar. 2. Open “Test” folder. 3. Click on “Upload” button on toolbar. 4. Select a file bigger than 2Gb on “Open” dialog and push “Open” button. | Selected file is uploaded to opened folder and size of this file is same as size of original local copy. |  |  |  |  |
| **8.** | **Uploading of large file/folder.** |  |  |  |  |  |
| 8.1. | 1. Click on “Upload” button on toolbar. 2. Select a large file (file should be bigger than 2Gb) on “Open” dialog and push “Open” button. | Selected file is uploaded. Compare sizes of uploaded file with original local copy and verify that local file is the same size as uploaded copy. |  |  |  |  |
| 8.2. | 1. Click on “Upload” button on toolbar. 2. Select few large files (every file should be bigger than 2Gb) on “Open” dialog and push “Open” button. | All selected files are uploaded. Compare sizes of uploaded files with original local copies and verify that all local files are the same size as uploaded copies. |  |  |  |  |
| 8.3. | 1. Click on “Upload Folder” on toolbar. 2. Select big folder (folder with size bigger than 2Gb) on “Browse folder” dialog and push “OK” button. | Selected folder is uploaded. Compare sizes of uploaded file with original local copy and verify that local folder is the same size as uploaded folder. |  |  |  |  |
| 8.4. | 1. Select few folders (every folder should have bigger than 2Gb size)on your file system. 2. Drop selected folders to list of files and folders of Ajax File Browser on web browser. | All selected folders are uploaded. Compare sizes of uploaded files with original local copies and verify that all local files are the same size as uploaded copies. |  |  |  |  |
| **9.** | **Uploading with using buttons of “Uploading Progress Panel”.** |  |  |  |  |  |
| 9.1. | 1. Click on “Upload” button on toolbar. 2. Select a large file (file should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Right click on opened file on “Uploading Progress Panel” 4. Choose “Pause” from context menu. 5. Right click on opened file on “Uploading Progress Panel” again. 6. Choose “Resume” from context menu. 7. Install “HashTab” application on your computer and compare checksum of uploaded file with original local file. | Uploading process is paused. After resuming file is uploaded correctly. Compare size of uploaded file with local copy and verify that they are the same. After comparing of checksum result should be correct. |  |  |  |  |
| 9.2. | 1. Click on “Upload” button on toolbar. 2. Select a large file (file should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Right click on opened file on “Uploading Progress Panel”. 4. Choose “Cancel” from context menu. | Uploading process is cancelled. File didn’t appear on list of files and folders of AjaxBrowser. |  |  |  |  |
| 9.3. | 1. Click on “Upload” button on toolbar. 2. Select few large files (files should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Select all files and right click on them on “Uploading Progress Panel”. 4. Choose “Pause” from context menu. 5. Right click on all opened files on “Uploading Progress Panel” again. 6. Choose “Resume” from context menu. | Uploading process is paused. After resuming all file are uploaded correctly. Compare sizes of uploaded files with local copies and verify that they are the same. |  |  |  |  |
| 9.4. | 1. Click on “Upload” button on toolbar. 2. Select few large files (files should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Select all files and right click on them on “Uploading Progress Panel”. 4. Choose “Cancel” from context menu. | Uploading process is cancelled. File didn’t appear on list of files and folders of AjaxBrowser. |  |  |  |  |
| 9.5. | 1. Click on “Upload” button on toolbar. 2. Select few large files (files should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Refresh web page of Ajax File Browser on web browser (by pushing on “Refresh” on browser or “F5” on keyboard). 4. Right click on any file on “Uploading Process Panel”. 5. Select “Resume” from context menu. 6. Choose any of opened on second step file again on “Open” dialog. | All uploading process should be paused after refreshing of Ajax File Browser page.  To resume uploading process it is needed to choose file again on “Open” dialog.  After reopening of file uploading should be continued from same percentage on which it was paused-not again from the beginning.  Verify that uploaded file is same size as original local file. |  |  |  |  |
| 9.6. | 1. Click on “Upload” button on toolbar. Select few large files (files should be bigger than 2Gb) on “Open” dialog and push “Open” button. 2. Refresh web page of Ajax File Browser on web browser (by pushing on “Refresh” on browser or “F5” on keyboard). 3. Right click on any file on “Uploading Process Panel”. 4. Select “Resume” from context menu. 5. Choose any other than on step 2 file on “Open” dialog. 6. Click “Close” on warning message dialog. | All uploading process should be paused after refreshing of Ajax File Browser page.  To resume uploading process it is needed to choose file again on “Open” dialog.  After reopening of file message “The following file(s) was modified…The uploading will start from the beginning of each file.” appears.  After closing of warning message dialog uploading of the file should start from the beginning.  Verify that file after uploading is same size as local copy of this file. |  |  |  |  |
| **10.** | **Checking on AjaxBrowser which was started by using "ASP.NET WebDAV Server Application Wizard" on Visual Studio.** |  |  |  |  |  |
| 10.1. | 1. Click on “Upload” button on toolbar. 2. Select a large file (file should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Compare checksum of uploaded file with by using original “HashTab” tab on properties of uploaded and local files. | Selected file is uploaded and has same size as original local file. After comparing of checksum result should be correct. |  |  |  |  |
| 10.2. | 1. Click on “Upload” button on toolbar. 2. Select few large files (every file should be bigger than 2Gb) on “Open” dialog and push “Open” button. | Selected files are uploaded and have same size as original local file. |  |  |  |  |
| 10.3. | 1. Click on “Upload Folder” on toolbar. 2. Select big folder (folder with size bigger than 2Gb) on “Browse folder” dialog and push “OK” button. | Selected folder is uploaded and has same size as original local file. |  |  |  |  |
| 10.4. | 1. Select few folders (every folder should have bigger than 2Gb size)on your file system. 2. Drop selected folders to list of files and folders of Ajax File Browser on web browser. | All selected folders are uploaded. Compare sizes of uploaded files with original local copies and verify that all local files are the same size as uploaded copies. |  |  |  |  |
| **11.** | **Checking on AjaxBrowser which was started by using "Add WebDAV Server Implementation Wizard" on Visual Studio.** |  |  |  |  |  |
| 11.1. | 1. Click on “Upload” button on toolbar. 2. Select a large file (file should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Compare checksum of uploaded file with by using original “HashTab” tab on properties of uploaded and local files. | Selected file is uploaded and has same size as original local file. After comparing of checksum result should be correct. |  |  |  |  |
| 11.2. | 1. Click on “Upload” button on toolbar. 2. Select few large files (every file should be bigger than 2Gb) on “Open” dialog and push “Open” button. | Selected files are uploaded and have same size as original local file. |  |  |  |  |
| 11.3. | 1. Click on “Upload Folder” on toolbar. 2. Select big folder (folder with size bigger than 2Gb) on “Browse folder” dialog and push “OK” button. | Selected folder is uploaded and has same size as original local file. |  |  |  |  |
| 11.4. | 1. Select few folders (every folder should have bigger than 2Gb size)on your file system. 2. Drop selected folders to list of files and folders of Ajax File Browser on web browser. | All selected folders are uploaded. Compare sizes of uploaded files with original local copies and verify that all local files are the same size as uploaded copies. |  |  |  |  |
| **12.** | **Checking on AjaxBrowser which was started by using "HttpListener WebDAV Server Application Wizard" on Visual Studio.** |  |  |  |  |  |
| 12.1. | 1. Click on “Upload” button on toolbar. 2. Select a large file (file should be bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Compare checksum of uploaded file with by using original “HashTab” tab on properties of uploaded and local files. | Selected file is uploaded and has same size as original local file. After comparing of checksum result should be correct. |  |  |  |  |
| 12.2. | 1. Click on “Upload” button on toolbar. 2. Select few large files (every file should be bigger than 2Gb) on “Open” dialog and push “Open” button. | Selected files are uploaded and have same size as original local file. |  |  |  |  |
| 12.3. | 1. Click on “Upload Folder” on toolbar. 2. Select big folder (folder with size bigger than 2Gb) on “Browse folder” dialog and push “OK” button. | Selected folder is uploaded and has same size as original local file. |  |  |  |  |
| 12.4. | 1. Select few folders (every folder should have bigger than 2Gb size)on your file system. 2. Drop selected folders to list of files and folders of Ajax File Browser on web browser. | All selected folders are uploaded. Compare sizes of uploaded files with original local copies and verify that all local files are the same size as uploaded copies. |  |  |  |  |
| **13.** | **Negative tests.** |  |  |  |  |  |
| 13.1. | Uploading with turned off Internet connection:   1. Load previously web page “www.ajaxbrowser.com” on web browser. 2. Click on “Upload” button on toolbar. 3. Select a file or create new one on “Open” dialog and push “Open” button. | Uploading isn’t starting (Error appears “Upload Progress Panel” and 10 seconds timeout before trying to reconnect). After turning on internet connection uploading restarts and finishes correct. |  |  |  |  |
| 13.2. | Turning off internet connection during uploading of the file:   1. Click on “Upload” button on toolbar. 2. Select a large file (file with size bigger than 2Gb) on “Open” dialog and push “Open” button. 3. Turn off internet connection on computer. | Uploading is pause on “Upload Progress Panel” and will start as soon as you turn on internet connection on your machine. |  |  |  |  |