Connecting Imagine to the Internet

This document outlines the set of procedures, which allow the developer to download/upload files from/to Internet using HTTP and FTP protocols as they were implemented in Imagine Logo during the EU project CoLabs (Socrates Minerva 2002-101301).

Internet-aware commands in Imagine Logo

The following new commands of the Main class implement the new functionality. Note that the names of these commands cannot be translated to local languages. They stay always in English.

`DownloadURL :URL :LocalPath`
`(DownloadURL :URL :LocalPath :params)`

Downloads a file or runs a script identified by the URL.
The URL is a word containing one fully qualified URL or a list of URLs. If it is a list, then the first item must be a word, which defines a fully qualified URL and the next items must define just file names relative to the fully qualified URL.
The fully qualified URL can be a HTTP or FTP one containing server name, username, password, port number, path and parameters (only server name and path is required), for example:
ftp://peter:hello@ftp.myserver.com/myfile.imp
http://myserver.com:8080/myscript.php?par1=1&par2=2
The filename of an FTP URL can include also wildcards * and ? in their usual meaning. Also elements of a :URL list can contain wildcards. HTTP URLs must not include wildcards.
The file(s) defined by the URL or the result of the script will be saved to file named by :LocalPath. If the :URL defines potentially more than one file (because it is a list or it is a word containing wildcards) then :LocalPath must be a name of a directory and all downloaded files will keep their file names. If the :URL names just one file then :LocalPath must can be either a file name or a directory name finished by a backslash character. In the later case the downloaded file's name will be the same as it was on the server.
FTP transfers of multiple files will establish one connection and transfer all files within that one connection.
Important: If the URL identifies a script including parameters then you should take into consideration that the rules of HTTP protocol define that such an invocation of a script should not result in permanent changes on the server. Invoking scripts, which result in such changes, should be done by the POST verb of HTTP protocol i.e. by using the PostRequest command as described below in this document.
The return value is a number defined by the HTTP protocol's status code.
The value 200 means OK, the value 404 means file not found. The complete list of status codes can be found in RFC2068 - http://www.ietf.org/rfc/rfc2068.txt or in Microsoft's description of the WinInet API - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wininet/wininet/http_status_codes.asp
Imagine adds special codes:
0 - LocalPath was empty (can happen only for UploadURL and PostFile). Using empty LocalPath is used for asking for a general permission for uploading files,

see the Client Security section for details.
1000 - An Internet session cannot be started,
1001 - Connection to a server cannot be established
1002 - I/O error when reading or writing the local file
The optional **:params** parameter is a list of pairs [paramName1 paramvalue1 paramName2 paramValue2 ...]. It can define values of parameters if they need to be different from default ones.
These parameters are implemented:
TransferType
The value can be **text**, **binary** or **default**. The default value is **default**.
This parameter controls the way how the file is transferred. There are two types of file transfers defined in the FTP protocol: text (ascii) and binary. In binary transfer type the file is transferred exactly with no changes. In text transfer type the file is assumed to be a plain text file according to the rules of the source computer and the file is transferred in such a way that it will become a valid plain text file on the destination computer. In praxis this means mostly that end-of-line markers are adjusted according to the type of operating system running on source and destination computers (e.g. Windows uses CRLF as end-of-line marker while Unix-based systems use just CR). These adjustments are done by the protocol itself, Imagine just asks for either type of transfer. For the HTTP protocol Imagine can just adjust incoming text files so that they comply to the WIndows rules for text files.
In case of **default** transfer type the actual transfer type is determined using the MIME type registered for the :LocalPath parameter's extension. If the Content-Type" defined for the particular file extension in the registry of the computer running Imagine starts with 'text/' then Imagine will use text transfer otherwise binary transfer.
PassiveFTP
The value can be **true** or **false**. The default value is **false**.
This parameter defines the way in which the FTP protocol will be used. It has no meaning for the HTTP protocol. Some FTP servers may not be able to transfer files using the "active" way. Then you can try using this flag to set the FTP transfer to be passive.

UploadURL :URL :LocalPath
(UploadURL :URL :LocalPath :ResponseFile)
(UploadURL :URL :LocalPath :ResponseFile :params)

Uploads one or more files using FTP or HTTP protocol. When executing this command Imagine asks the user for permission to upload the file unless the server which is being uploaded to has a general permission (see the Client Security section for details).
The :URL can be of any form described for the DownloadURL command but it must not contain parameters or wildcards and it must be just one word. This URL identifies the file on an FTP or HTTP server or a directory on an FTP or HTTP server (if its last character is a slash or if :LocalPath contains wildcards or :LocalPath is a list).
:LocalPath is either a word or a list and its file name can include wildcards * and ? in their usual meaning. If it is a list then the first element must be a fully qualified path while the rest of the list should contain paths relative to the first element.
If the URL is an FTP one then uploading will happen using the FTP protocol. If more than one file is transferred then this will happen within one FTP connection. It means that the developer, who uses this command, must know an FTP site where he knows a user name and password, which has permissions to create and write files.
If the URL is an HTTP one then uploading will happen using the PUT verb of HTTP

protocol. This verb must be allowed on the server (perhaps for a specific user). Usually the PUT method is not allowed. Therefore uploading with HTTP protocol can be useful only in situations where the developer controls the Web server and is able to set up the PUT verb to work safely. For the HTTP protocol PostFile or PostRequest is a more useful method of uploading files.
The return value is a number defined by the HTTP protocol's status code as described in DownloadURL.
The optional **:ResponseFile** parameter has a meaning only for the HTTP protocol. When uploading a file using HTTP then the server not only gives a status code, but it also returns some data (which is usually an HTML file) - the HTTP response. If :ResultFile is given then Imagine stores the response into the file, which name is in the :ResponseFile parameter. If the parameter is not present or if its value is an empty string or empty list then the response is not stored at all.
The optional **:params** parameter has the same meaning as it is explained for the DownloadURL command.

PostFile :URL :LocalPath :ResponseFile (PostFile :URL :LocalPath) (PostFile :URL :LocalPath :ResponseFile :params)

Posts a file using the POST verb of the HTTP protocol. When executing this command Imagine asks the user for permission to post the file unless the scrip which is being posted to has a general permission (see the Client Security section for details).
It simulates the action performed by a browser when posting the information for this HTML form (where URL is the one given as the first input):
```
<form name="PostForm" action="URL" method="Post"
enctype="multipart/form-data">
<input type="file" class="textbox" name="file" value="" size="60">
<input type="submit" name="action" value="Upload">
</form>
```
The **:URL** must identify a server side script (for example a PHP script), which is able to get the file and store it at a location on the server.
This is an example of a simplest PHP script (written by Tiago), which can handle the upload on the server:
```
<script language="php">
  if ($HTTP_POST_VARS["action"] == "Upload")
  {
    $filename = $HTTP_POST_FILES["file"]["name"];
    $destination_file = "files/" . $filename;
    if (copy ($HTTP_POST_FILES["file"]["tmp_name"],$destination_file))
    {
     echo "<p><b>File uploaded</b></p>";
    }
    else
    {
     echo "<p><b>Error uploading.</b></p>";
    }
  }
</script>
```
Note that the script does not check any details of the file, just puts it to the files subfolder of the folder where the script itself is located. If that folder can be accessed using HTTP protocol then the script may be more complicated to check for allowed extensions not to allow uploading malicious scripts to the server.

For downloading files uploaded by PostFile the developer can use either DownloadURL with a known URL of the file (if the script on the server will put the file on a place, which is accessible by HTTP or FTP) or DownloadURL defining a script's address, which will take care of downloading the file.
The return value is a number defined by the HTTP protocol's status code as described in DownloadURL. **Important:** If the script does not define the status code then the code 200 can mean that the request was properly delivered to the script on the server, not that the script actually successfully used the request. In most scripting languages there is a way in which the script can determine the status code, which is then sent to the client. We recommend using this feature of scripts to inform the Imagine program properly about what happened with the post request after the script has processed it.
The **:ResponseFile** parameter gives the file name where to store the response to. The response is the result of the running script. For scripts, which were developed for interacting directly with a user using a browser, the response is usually an HTML file, which informs the user if the file was uploaded successfully or not. For scripts developed specifically for use with Imagine projects we recommend that the result should be just a simple text string, which can be then easily read by an Imagine program and check what was the result of posting the file. If **:ResponseFile** is an empty word or an empty list then the response is not stored at all.
The optional **:params** parameter is a list of pairs [paramName1 paramvalue1 paramName2 paramValue2 ...]. It can define values of parameters if they need to be different from default ones.
These parameters are implemented:
TransferType
it is described in the DownloadURL command
MIMEtype
Defines the MIME type of the file, which will be reported to the HTTP server. If it is not set then the MIME type is determined from the registry of the client computer.

PostRequest :URL :Request :ResponseFile
(PostRequest :URL :Request)
(PostRequest :URL :Request :ResponseFile :params)

Posts a complete request using the POST verb of the HTTP protocol. When executing this command if there are files uploaded by the request then Imagine asks the user for permission to post each such file unless the script, which is being posted to, has a general permission (see the Client Security section for details).
The **:URL** must identify a server side script (for example a PHP script), which is able to perform an action based on this request.
:Request is a list describing the HTTP request sent to the script. HTTP requests are posted usually as a reaction of a browser to a form defined by an HTML page. You can get more insight about how the FORM tag of HTML relates to HTTP requests by reading the description of HTML.
Technically the request (even if it is a result of filling in a complicated HTML form) is just a series of (name, value) pairs, which are transferred from the client to the server. There can be two kinds of pairs in one of them the value is directly a string in others the value is just a local file name and the actual value sent to the server is the content of that file. The script can read all the names and values using a mechanism specific to a particular scripting language. For PHP scripts you can find the details here:
http://www.php.net/manual/en/features.file-upload.php

http://www.php.net/manual/en/
In Imagine the **:Request** parameter is a list in the form:
[name1 value1 name2 value2 ... nameN valueN]
or
[name1 value1 name2 value2 ... nameN valueN encodingType]
nameX elements are words, they will become the name part of the (name,value) pair sent to the server.
valueX elements are either words (then the word's value is sent to the server as the value) or lists in the form [localFileName] or [localFileName mimeType]. In both cases the content of the file referred by localFileName is sent to the server. If there is no mimeType (i.e. the list has just one element) then the mime type of the file reported to the server is determined from the registry of the client computer according to the extension of localFileName. If the registry defines no MIME type for that extension then it is set to application/octet-stream (which means general purpose file with no idea about its internals). If the mimeType is given then it must be a word and its value defines the MIME type. MIME types are discussed in http://www.ietf.org/rfc/rfc1521.txt (maybe there is a newer RFC on this topic, which defines more types, I am not sure).
encodingType - if present then it must be the last and odd element of the request list. It must be a word with either of these two values: application/x-www-form-urlencoded or multipart/form-data. If there are files uploaded by the request then the encoding is set always to multipart/form-data. If there are no files uploaded and no encoding type is given then it defaults to application/x-www-form-urlencoded.
Many scripts get the request in a preprocessed form (depending on the scripting language and tools used to write the script), so the script itself does not need to deal with encoding type anyway and therefore you may leave it at default values. Change it only if you know that it is needed.
application/x-www-form-urlencoded means that the (name,value) pairs will be sent to the server encoded in the same way as in GET requests (e.g. [n1 v1 n2 v2] will be encoded as a string n1=v1&n2=v2 with some special rules about values containing special characters).
multipart/form-data uses an upload type defined by RFC 1867 - see http://www.ietf.org/rfc/rfc1867.txt for details.

lastContentType

This function reports the MIME type of the last HTTP transfer.
After executing DownloadURL from an HTTP URL, UploadURL to an HTTP url, PostFile or PostRequest you can ask for lastCOntentType to know what type of information was the server sending to your client. This information can be useful in situations when the type of information cannot be deduced from the URL itself e.g. when you invoke a script. For example your program can try to download a page identified by a PHP script's address:
DownloadURL "|http://www.myaddress.sk/myscript.php?par1=1&par2=45| "|c:\localfile.htm|
The response of the script is store to c:\localfile.htm without knowing if it was really an html code or any other kind of information.
After executing the above command we can look at lastContentType to verify the content type. If it is text/html, then it is OK, but if it is image/jpeg, then we need to rename the file probably before trying to load it into an Imagine program.

transferStatus :processname

This function allows reporting status information for long downloads or uploads.
Its input is a name of a process (usually an explicit name given to it by the optional input of **launch**, **forever** and **every** commands).
If a process executes a DownloadURL, UploadURL, PostFile or PostRequest command then its execution will go on only after the command has been completed. But the command may take a long time if the files being transferred are big. In such cases the developer may run the command in one process while other processes may ask for the command's status using the transferStatus function and may display the progress to the user or may even stop the process using the **cancel** command..
Its output is a list of three elements:
[direction totalbytes currentbytes]
direction is either an empty word or the word **up** or the word **down**. It indicates the direction of the transfer. The empty word indicates that in the particular process there is no running Internet transfer.
totalbytes indicate the total length of the current transfer to be completed.
currentbytes indicate the amount of bytes actually transferred.
Note that **UploadURL** for HTTP addresses, **PostFile** and **PostRequest** include two transfers (unless the **:responseFile** parameter is empty). The first transfer is **up** and it delivers a file or an HTTP request to the server and then a **down** transfer follows it to deliver the response to the client. In such a case **totalbytes** refers to the total number of bytes for the up or down transfer not the total number of bytes for the whole command (it is impossible to know the size of the response before the request was completely sent).

FTPOpen :FTPURL
(FTPOpen :FTPURL :params)

Opens an FTP connection. It means that all subsequent FTP transfers or requests to the same server using the same port, username and :params, which occur in the same Logo process will re-use the same connection, which will make them much faster compared to the situation when a sequence of FTP transfers would be executed without FTPOpen. Each connection opened by FTPOpen must be closed by and FTPClose. FTPCLose is issued automatically when a Logo process is finishing.
The user must be aware of the fact that some FTP servers may close the connection themselves if it is not active for some time. In such cases some transfers, shich use the connection may fail. Therefore the user should not leave a connection open for a long time with any transfer.

FTPClose :FTPURL
(FTPClose :FTPURL :params)

Closes a connection opened by FTPOpen.

erasefile :FTPURL
erasefile? :FTPURL

These two imagine commands were now extended in two directions: They can use an FTP URL instead of a local file name and also they can specify more than one file using either a list (using the same rules as downloadURL and UploadURL) or wildcards.

filelist :FTPURL
folderlist :FTPURL
file? :FTPURL

folder? :FTPURL
renamefile :FTPURL :NewName
renamefile? :FTPURL :NewName
erasefolder :FTPURL
erasefolder? :FTPURL
createfolder :FTPURL
createfolder? :FTPURL
All these Imagine procedures were now extended so that they can use an FTP URL instead of a local file or folder name.

Client Security

When uploading local files to Internet servers there is a security hazard. A malicious Imagine program could upload sensitive information from the local computer to any server.
Therefore some security measures must be taken.
The current measures were derived from how Internet Explorer handles POST requests issued by a script (i.e. not as a reaction to clicking Submit button in the form by the user). In such a case IE asks the user if he/she allows the script to upload the file.
In Imagine we implemented a similar mechanism. Its basic rule says that whenever a file is being uploaded or posted using UploadURL, PostFile or PostRequest commands, Imagine asks the user for permission. As this may be annoying for applications uploading several files, there are some shortcuts:

1. The dialog box of the question contains a checkbox, which allows the user to grant a general permission for one server (in UploadURL) or script (in PostFile or PostRequest).
2. UploadURL and PostFile can be called with an empty **:LocalPath** parameter (i.e. it is an empty list or an empty word). Then Imagine asks for the general permission for the server and does not perform any uploading operation. Such command can be put to the beginning of an Imagine program so the user can decide at the beginning that he/she allows uploading files to a particular server or script and then he/she needs not confirm each particular file later. When working with small children the teacher can handle the dialog for the children (clicking Yes in it) just at the beginning and then the children will not be distracted by any questions later.

The implementation of security measures described above is given for discussion among the project partners. In future we may change them based on the results of the discussion. The dialog shown to the user is a preliminary one, after the discussion we may make its more readable.

Server Security

There are security issues also on the server side.
The developer of server side scripts should think carefully about avoiding security problems when uploading files. The most obvious danger is that a user could upload a script to the server and the execute it. To avoid or minimize the risk the script used for posting a file should not allow uploading any kinds of files, just specific ones needed for the particular purpose. The may be protected by a username and password either inside the HTTP protocol (it is possible to use URLs, which include username and password

using the usual notation like http://username:password@myserver.com/abc.php) or as an additional parameter in a posted request. Also the whole upload directory may be protected for HTTP and FTP accesses.

Current state

The commands described above were implemented in a test version of Imagine 325, which English version is being distributed to Colabs Partners along with this document. Translation of the commands and associated dialogs is not yet available, it will be possible just after the definitive set of commands is agreed based on discussions with project partners.
We have tested all functionality using an Apache server on Windows XP in our Intranet environment and also on an external Linux server.
New functions and changes marked by violet colour were added in build 330 and distributed to all CoLabs partners.