iControl 101 - #06 - File Transfer APIs

Joe Pruitt, 2008-10-01

The main use we see for iControl applications is with the automation of control features such as adding, removing, enabling, and disabling objects.   Another key step in multi-device management is the automation of applying hotfixes and other software updates as well as the downloading of configurations for archival and disaster recovery purposes. iControl has a set of methods that enable the uploading and downloading of files for these purposes. This article will discuss these "file transfer" APIs and how you can use them for various management purposes.

The File Transfer APIs

The API methods uses to transfer files to and from the device can be found in the System::ConfigSync interface. You may ask: Why are they ConfigSync interface? The answer is quite simple actually. In our first version of iControl, we had the need to transfer configurations across devices in a HA pair as part of the Configuration Sync process. So in doing so, we introduced the upload_configuration() and download_configuration() methods. When we later added more generic file transfer APIs, it seemed logical to place them next to the pre-existing configuration transfer methods. So, that's the reason...

The following methods are used to download content:

```
FileTransferContext System::ConfigSync::download_configuration(
    in String config_name,
    in long chunk_size,
    inout long file_offset
);
FileTransferContext System::ConfigSync::download_file(
    in String file_name,
    in long chunk_size,
    inout long file_offset
);
```

And the following two methods are used to upload content:

```
void System::ConfigSync::upload_configuration(
    in String config_name,
    in System::ConfigSync::FileTransferContext file_context
);
void System::ConfigSync::upload_file(
    in String file_name,
    in System::ConfigSync::FileTransferContext file_context
);
```

The above methods use the following enum and structure as part of the control

```
enum Common::FileChainType {
    FILE_UNDEFINED = 0
    FILE_FIRST = 1,
    FILE_MIDDLE = 2,
    FILE_UNUSED = 3,
    FILE_LAST = 4,
    FILE_FIRST_AND_LAST
};
struct System.::ConfigSync::FileTransferContext {
    char [] file_data,
    Common::FileChainType chain_type
};
```

Chunks

Due to the limitations with SOAP's with regards to payload and processing of large messages, we designed the file transfer APIs to work in "chunks". This means that for a multi-megabyte file, you a loop in your code to send up "chunks" in whatever chunk sizes you wish from 1 byte to 100's of Ks. We recommend not getting too extreme on either end of the chunk size ranges. Typically we recommend 64 to 256k per chunk.

How to use the download methods.

The process for downloading content is fairly straightforward. In this example, we'll use the download_configuration method.

1. Make a call to the download_configuration method with a given configuration name (a list of existing configurations can be returned from ConfigSync::get_configuration_list) with the requested chunk_size (ie 64k) and the starting file_offset of 0.
2. The first response will come back in the FileTransferContext with the data and the FileChainType describing whether this was the first chunk, a middle chunk, the last chunk, or the first and last chunk.
3. take the encoded data stored in the FileTransferContext.file_data array and save it locally.
4. If the FileChainType is FILE_LAST or FILE_FIRST_AND_LAST, you are done.
5. Otherwise, use the incremented file_offset go to step #1.

The following snippet of code taken from the iControl SDK's ConfigSync C# sample application illustrates how to deal with downloading a file with an unknown size.

```
void handle_download(string config_name, string local_file)
{
    ConfigSync.SystemConfigSyncFileTransferContext ctx;
    long chunk_size = (64*1024);
    long file_offset = 0;
    bool bContinue = true;
    FileMode fm = FileMode.CreateNew;
    if ( File.Exists(local_file) )
    {
        fm = FileMode.Truncate;
    }
    FileStream fs = new FileStream(local_file, fm);
    BinaryWriter w = new BinaryWriter(fs);
    while ( bContinue )
    {
        ctx = ConfigSync.download_configuration(config_name, chunk_size, ref file_offset);
        // Append data to file
        w.Write(ctx.file_data, 0, ctx.file_data.Length);
        Console.WriteLine("Bytes Transferred: " + file_offset);
        if ( (CommonFileChainType.FILE_LAST == ctx.chain_type) ||
            (CommonFileChainType.FILE_FIRST_AND_LAST == ctx.chain_type) )
        {
            bContinue = false;
        }
    }
    }
    w.Close();
}
```

How to use the upload methods

The upload methods work in a similar way but in the opposite direction. To use the upload_configuration configuration method, you would use the following logic.

1. Determine the chunk size you are going to use (64k - 256k recommended) and fill the file_data array with the first chunk of data.
2. Set the FileChainType to FILE_TYPE_FIRST_AND_LAST if all your data can fit in the first chunk size.
3. Set the FileChainType to FILE_FIRST if you fill up the data with your chunk size and there is more data to follow.
4. Make a call to upload_configuration with the configuration name and the FileTransferContext with the data and the FileChainType.
5. If the data has all been sent, stop processing.
6. Else if the remaining data will able to fit in the given chunk size, set the FileChainType to FILE_LAST, otherwise set it to FILE_MIDDLE.
7. Fill the file_data with the next chunk of data.
8. Goto Step #4.

The following example taken from the iControl SDK ConfigSync perl sample and illustrates taking a local configuration file and uploading it to the device.

```
sub uploadConfiguration()
{
    my ($localFile, $configName) = (@_);
    $success = 0;
    $bContinue = 1;
    $chain_type = $FILE_FIRST;
    $preferred_chunk_size = 65536;
    $chunk_size = 65536;
    $total_bytes = 0;
    open(LOCAL_FILE, "<$localFile") or die("Can't open $localFile for input: $!");
    binmode(LOCAL_FILE);
    while (1 == $bContinue )
    {
        $file_data = "";
        $bytes_read = read(LOCAL_FILE, $file_data, $chunk_size);
        if ( $preferred_chunk_size != $bytes_read )
        {
            if ( $total_bytes == 0 )
            {
                $chain_type = $FILE_FIRST_AND_LAST;
            }
            else
            {
                $chain_type = $FILE_LAST;
            }
            $bContinue = 0;
        }
        $total_bytes += $bytes_read;
        $FileTransferContext =
        {
            file_data => SOAP::Data->type(base64 => $file_data),
            chain_type => $chain_type
        };
        $soap_response = $ConfigSync->upload_configuration
        (
            SOAP::Data->name(config_name => $configName),
            SOAP::Data->name(file_context => $FileTransferContext)
        );
        if ( $soap_response->fault )
        {
            print $soap_response->faultcode, " ", $soap_response->faultstring, "\n";
            $success = 0;
            $bContinue = 0;
        }
        else
        {
            print "Uploaded $total_bytes bytes\n";
            $success = 1;
        }
        $chain_type = $FILE_MIDDLE;
    }
    print "\n";
    close(LOCAL_FILE);
    return $success;
}
```

Other methods of data transfer

The two methods illustrated above are specific to system configurations. The more generic upload_file() and download_file() commands may be used to do things like backing up other system files as well as, but not limited to, uploading hotfixes to be later installed with the System::SoftwareManagement::install_hotfix() method. The usage of those methods is identical to the configuration transfer methods except in the fact that the config_name parameter is now replaced with a fully qualified file system name. Note that there are some restrictions as to the file system locations that are readable and writable. You wouldn't want to accidentally overwrite the system kernel with your latest hotfix would you?

Conclusion

Hopefully this article gave you some insights on how to use the various file transfer APIs to manipulate file system content on your F5 devices.

Get the Flash Player to see this player.