Mi scuso per l'inconveniente; il link generato nell'ambiente di lavoro protetto (sandbox) a volte può avere problemi di accesso diretto a seconda delle impostazioni del browser.

Dato che non posso caricare file direttamente sul tuo Drive o inviarli via email, la soluzione migliore per assicurarmi che tu abbia il contenuto è **fornirti qui sotto il testo integrale tradotto in inglese**. Puoi copiarlo e incollarlo in un nuovo file Word (o Google Doc) sul tuo computer.

Ecco la traduzione completa e professionale del manuale [1]:

User Manual and Scopes of Use for ntJobApp

Version 20251226 [2]

1. Introduction and Purpose

An **ntJobApp** is an application designed to operate in **batch mode**[3]. The architecture utilizes the ncJobApp class and the jData instance to orchestrate these batch micro-applications[4]. The version generated via AI uses the acJobsOS class and the aiSys function file[5].

The purpose of an ntJobApp is threefold[6]:

- Read an .ini file passed as a parameter (usually named ntjobs.ini)[7].
- Execute one or more commands requested by the caller[8].
- Return a result .ini file. The output .ini file will be renamed with the .end extension[9].

ntJobApps are built to be orchestrated by **ntJobsOS**, but they can also function autonomously[10].

2. Execution Logic for the Executor

The execution of an ntJobApp follows a standard sequential logic[11]:

1. **Instance Creation**: The ncJobApp instance is created in the jData global variable[12].

2. **Startup (Start())**: self.Start() is executed, which reads the .ini file passed as a parameter (or creates it) and saves it in self.dictJobs[13]. If sResult (the operation result) is not empty, the application exits with **errorlevel 1**[14].
3. **Execution (Run())**: self.Run() is executed, running each individual job defined in self.dictJobs in sequence[15].
4. **Finalization (End())**: self.End(sResult) is executed[16].

3. Execution Instructions

To execute an ntJobApp, the application must be started by passing the **full path** of the .ini configuration file as the first parameter[17].

3.1. The Input File (.ini)

The .ini file must contain at least the [CONFIG] section and one or more **job** sections[18].

Example ntjobsapp.ini file: [19]

```
Ini, TOML

[CONFIG]
TYPE=NTJOBS.APP.1.0
EXIT=TRUE       ; If "TRUE", the app exits if a job fails.
LOG="FILE.LOG"  ; Optional Log file name.
NAME=APP_ID

[JOB1_ID]
ACTION=ACTION_NAME
FILE.ID1=PathFile1
PARAM.ID1=Value
```

Crucial Validations in Start():

- **CONFIG Section**: Must be present in the dictionary loaded into self.dictJobs, otherwise the app exits with the error "CONFIG section not found"[20].
- **COMMAND Key**: Every job section (except CONFIG) must have a COMMAND key (or ACTION in the example); otherwise, the error is appended to sResult[21].

- **File Verification**: If there are keys starting with FILE., the application verifies that the file exists[22]. If a file does not exist, the error is appended to sResult, but execution is **not** interrupted at this stage[23].

3.2. The Output File (.end)

At the end of execution, the End(sResult) method saves the content of self.dictJobs as an .ini file at the path self.sJobEnd[24]. self.sJobEnd is the name of self.sJobIni with the extension changed to .end[25].

The .end file will contain the results of each job, including[26]:

- RETURN.TYPE (e.g., E=Error, ""=OK)[27].
- RETURN.VALUE (Optional return message)[28].
- TS.START and TS.END (Start and end timestamps)[29].
- Keys with the prefix FILE.RETURN. (references to files generated by the job)[30].

3.3. Result Codes (Errorlevel)

The ntJobApp returns a result code (errorlevel) indicating the final status[31]:

Code	Description	Meaning
0	All Good	The application finished correctly; all jobs were executed.[32]
1	Failed to reach job execution	Critical error during the startup phase (Start()); the .end file is not generated.[33]

2	One or more jobs failed	One or more jobs within the .ini file ended with an error.[34]

4. Scopes of Use and Advantages vs. Web Automation

The ntJobApp architecture is specifically designed for **batch** automation processes and offers significant advantages over solutions requiring graphical interaction (like clicks and waits on a website)[35].

Typical ntJobApp Use Cases	Key Advantages vs. Web Automation (GUI)
Asynchronous Processes: Complex or massive processing, such as nightly report generation, that should not block the user.[36]	**Efficiency and Speed**: As a **backend** process, it has no graphical overhead and doesn't require waiting for page rendering or human interaction.[37]
Data Integration: Massive file movement, conversion, or processing (e.g., FTP to a database) working "under the hood."[38]	**Robustness and Reliability**: Not subject to errors from UI changes (like a button changing ID) or network timeouts during page loads.[39]
Structured Orchestration: Calling different applications sequentially, where results/files from one action are inputs for the next.[40]	**Standardized Traceability**: The .ini input and .end output ensure a uniform, readable status format for the caller.[41]

Scheduling: Executing automatic tasks at predefined times (e.g., Cron or Task Scheduler).[42]	**Ease of Automation**: Command-line execution makes it ideal for any **job scheduler**.[43]

Special Features of NTJOBSAPP and NTJOBSOS

An NTJOBSAPP must be simple, portable, and focused on automation and asynchronous communication[44].

- **Similarities with Docker**: Like a Docker image composition using YAML files to tell the composer what to do in sequence, ntJobApp uses .ini files[45].
- **Similarities with Microservices**: Applications are broken down into autonomous, reusable modules[46]. An ntJobsApp is composed of modules ("commands") independently callable by the user[47].
- **Sustainability**: The core paradigm is sustainability, which involves trade-offs like the absence of a WEB interface[48]. The user interface uses "simple" channels and execution is asynchronous[49].

IT Structure

- **Portability**: ntJobsApp and ntJobsOS must be easily movable and low-cost to maintain[50]. They require a Python X64 portable environment[51].
- **Simplicity**: No mandatory underlying databases or web server installations are used[52]. Configuration is handled via .csv and .ini files[53].
- **Components**:
  - aiSys.py or nlSys.py: Generic function/class library[54].
  - acJobsApp.py or ncJobsApp.py: Interface and management class[55].
  - Start function for initialization and cbCommands() callback for executing commands[56].

Polling Events (Similarities with IBM MQ / Apache NiFi)

In this mode:

1. The user/app deposits a file in an input folder[57].
2. A daemon/service checks the folder regularly[58].
3. New files are validated, moved to a work folder, and the job is enqueued/executed[59].
4. The worker executes the guided flow (e.g., host program calls, scripts)[60].
5. Results are output to a folder or MQ[61].

This pattern is currently being tested for integration with **Apache NiFi** [62].

Puoi copiare questo testo in un documento Word e salvarlo. Se hai bisogno di modifiche specifiche alla traduzione, fammelo sapere!