Wild Apricot API Technical Guide

Overview

This document provides a comprehensive technical reference for how the SBNC Chatbot system integrates with the Wild Apricot (WA) API to synchronize membership data.

---

1. API Credentials and Configuration

1.1 Configuration File Location

Server path: `/home/sbnewcom/wa_sync/wa_config.json`

```
{
  "wildapricot": {
    "api_key": "<API_KEY>",
    "account_id": "176353"
  },
  "database_path": "data/wa.db",
  "log_file": "logs/wa_sync.log",
  "log_level": "INFO"
}
```

1.2 Credential Components

Component	Description	Location
`api_key`	Wild Apricot API key (from WA Admin > Settings > Integrations > Authorized applications)	`wa_config.json`
`account_id`	Your Wild Apricot account ID (visible in WA admin URL)	`wa_config.json`

1.3 Security Notes

- The API key has full read access to all account data
- Never commit `wa_config.json` to version control
- The file should be readable only by the sync user (`chmod 600`)

---

2. Authentication Flow

2.1 OAuth 2.0 Client Credentials Grant

Wild Apricot uses OAuth 2.0 with the **client_credentials** grant type. The API key acts as the client secret.

2.2 Token Request

Endpoint: `https://oauth.wildapricot.org/auth/token`

Method: POST

Headers:
`Authorization: Basic <base64(APIKEY:{api_key})> Content-Type: application/x-www-form-urlencoded`

Body: `grant_type=client_credentials&scope=auto`

2.3 Sample Authentication Code

```python
import base64
import requests
import time

class WildApricotAuth:
    def __init__(self, api_key: str):
        self.api_key = api_key
        self.token_cache = None
        self.token_expiry = None

    def get_access_token(self) -> str:
        """Obtain OAuth access token from Wild Apricot."""

        # Return cached token if still valid
        if self.token_cache and self.token_expiry and time.time() < self.token_expiry:
            return self.token_cache

        # Build Basic auth header: base64("APIKEY:{api_key}")
        credentials = base64.b64encode(f"APIKEY:{self.api_key}".encode()).decode()

        headers = {
            "Authorization": f"Basic {credentials}",
            "Content-Type": "application/x-www-form-urlencoded"
        }

        data = {
            "grant_type": "client_credentials",
            "scope": "auto"
        }

        response = requests.post(
            "https://oauth.wildapricot.org/auth/token",
            headers=headers,
            data=data
        )
        response.raise_for_status()

        token_data = response.json()
        self.token_cache = token_data['access_token']

        # Set expiry to 5 minutes before actual expiry for safety
        expires_in = token_data.get('expires_in', 1800) - 300
        self.token_expiry = time.time() + expires_in

        return self.token_cache
```

2.4 Token Response

```
{
  "access_token": "eyJ0eXAiOiJKV1Q...",
  "token_type": "Bearer",
  "expires_in": 1800,
  "refresh_token": "...",
  "Permissions": [
    {"AccountId": 176353, "SecurityProfileId": 1}
  ]
}
```

2.5 Token Lifecycle

- Tokens expire after **30 minutes** (1800 seconds)
- Our code caches tokens and refreshes 5 minutes before expiry
- No refresh token logic needed - just request a new token

---

3. API Request Structure

3.1 Base URL

```
https://api.wildapricot.org/v2.2/accounts/{account_id}/{endpoint}
```

For our account: `https://api.wildapricot.org/v2.2/accounts/176353/`

3.2 Request Headers

```
Authorization: Bearer {access_token}
Accept: application/json
Content-Type: application/json
```

3.3 Sample API Request Code

```
def make_api_request(self, method: str, endpoint: str,
                     params: dict = None, data: dict = None) -> dict:
    """Make authenticated API request to Wild Apricot."""

    token = self.get_access_token()
    url = f"https://api.wildapricot.org/v2.2/accounts/{self.account_id}/{endpoint}"

    headers = {
        'Authorization': f'Bearer {token}',
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    }

    response = requests.request(
        method=method,
        url=url,
        headers=headers,
        params=params,
        json=data,
        timeout=30
    )
    response.raise_for_status()
    return response.json()
```

4. Data Retrieval Patterns

4.1 Standard Pagination

Most endpoints support OData-style pagination:

Parameter	Description
`$top`	Number of records to return (max 500)
`$skip`	Number of records to skip
`$filter`	OData filter expression

Sample paginated fetch:

```
def fetch_paginated(self, endpoint: str, page_size: int = 100) -> list:
    """Fetch all pages from a paginated endpoint."""
    all_items = []
    skip = 0

    while True:
        params = {
            '$top': page_size,
            '$skip': skip
        }

        response = self.make_api_request('GET', endpoint, params=params)

        # Handle different response formats
        items = response.get('Items', response.get('Events',
                response.get('Invoices', [])))

        if not items:
            break

        all_items.extend(items)

        # If we got fewer than page_size, we've reached the end
        if len(items) < page_size:
            break

        skip += page_size

    return all_items
```

4.2 Asynchronous Queries (Contacts)

The Contacts endpoint uses **async queries** for large datasets. Instead of returning data directly, it returns a `ResultUrl` that must be polled.

Step 1: Initiate async query

```
# Request contacts with async flag
params = {'$async': 'true'}
response = self.make_api_request('GET', 'contacts', params=params)

# Response contains ResultUrl, not data
# {"ResultId": "...", "ResultUrl": "https://api.wildapricot.org/v2.2/..."}
```

Step 2: Poll for results

```
def poll_async_result(self, result_url: str, max_attempts: int = 40) -> list:
    """Poll async API result until complete."""

    token = self.get_access_token()

    for attempt in range(max_attempts):
        time.sleep(3)  # Wait 3 seconds between polls

        response = requests.get(
            result_url,
            headers={
                'Authorization': f'Bearer {token}',
                'Accept': 'application/json'
            },
            timeout=30
        )
        response.raise_for_status()
        result = response.json()

        state = result.get('State')

        if state == 'Complete':
            return result.get('Contacts', result.get('Items', []))
        elif state == 'Failed':
            error = result.get('ErrorDetails', 'Unknown error')
            raise Exception(f"Async query failed: {error}")
        elif state in ['Waiting', 'Processing']:
            continue  # Keep polling

    raise TimeoutError("Async query did not complete in time")
```

4.3 Incremental Sync with Filters

For daily incremental updates, use the `$filter` parameter:

```
# Fetch contacts modified in last 24 hours
from datetime import datetime, timedelta

yesterday = (datetime.now() - timedelta(days=1)).strftime('%Y-%m-%d')
filter_param = f"'Profile last updated' gt {yesterday}"

params = {
    '$async': 'true',
    '$filter': filter_param
}

response = self.make_api_request('GET', 'contacts', params=params)
```

5. API Endpoints Used

5.1 Contacts (Members)

Endpoint: `contacts`

Method: GET

Special handling: Uses async queries

Response fields used: - `Id` - Unique contact ID - `FirstName`, `LastName`, `Email` - `MembershipLevel.Id`, `MembershipLevel.Name` - `Status` - Active, Lapsed, PendingNew, etc. - `MemberSince` - Join date - `FieldValues` - Custom fields (JSON array) - `ProfileLastUpdated` - Last modification timestamp

5.2 Events

Endpoint: `events`

Method: GET

Uses standard pagination

Response fields used: - `Id`, `Name`, `StartDate`, `EndDate` - `Location` - `Organizer.Id` - Contact ID of organizer - `RegistrationEnabled`, `RegistrationsLimit` - `ConfirmedRegistrationsCount`, `PendingRegistrationsCount` - `AccessLevel` - Public, Members only, etc. - `Tags` - Array of tag strings

5.3 Event Registrations

Endpoint: `eventregistrations?eventId={event_id}`

Method: GET

Note: Must be fetched per-event

```
def fetch_event_registrations(self, events: list) -> list:
    """Fetch registrations for all events."""
    all_registrations = []

    for event in events:
        event_id = event.get('Id')
        params = {'eventId': event_id}

        response = self.make_api_request('GET', 'eventregistrations', params=params)

        # Handle different response formats
        if isinstance(response, list):
            regs = response
        else:
            regs = response.get('EventRegistrations',
                   response.get('Items', []))

        all_registrations.extend(regs)

    return all_registrations
```

5.4 Invoices

Endpoint: `invoices`

Method: GET

Uses standard pagination

5.5 Membership Levels

Endpoint: `membershiplevels`

Method: GET

No pagination needed (typically < 10 records)

5.6 Email Tracking

Endpoint: `SentEmails`

Method: GET

Supports filtering by date:

```
params = {
    '$filter': f"SentDate ge {start_date}",
    '$top': 200
}
```

Recipient details:

Endpoint: `SentEmailRecipients?emailId={email_id}`

---

6. Server File Locations

6.1 Sync Scripts

File	Location	Purpose
`wa_full_sync.py`	`/home/sbnewcom/wa_sync/`	Full data synchronization
`wa_config.json`	`/home/sbnewcom/wa_sync/`	API credentials
`wa_db_optimizer.py`	`/home/sbnewcom/wa_sync/`	Creates views/indexes after sync

6.2 Cron Schedule

```
# Daily incremental sync at 3:00 AM
0 3 * * * cd /home/sbnewcom/wa_python && ./run_incremental_sync_with_retry.sh

# Weekly full sync on Sundays at 2:00 AM
0 2 * * 0 cd /home/sbnewcom/wa_sync && ./run_full_sync_with_retry.sh

# Health check every 6 hours
0 */6 * * * /root/scripts/check_wa_sync_health.sh
```

6.3 Database Location

- **Primary:** `/home/sbnewcom/wa_sync/data/wa.db`
- **Chatbot symlink:** `/var/www/chatbot/backend/wa.db` -> `/home/sbnewcom/wa_sync/data/wa.db`

---

7. Error Handling

7.1 Common HTTP Errors

Code	Meaning	Action
401	Token expired	Refresh token and retry
429	Rate limited	Wait and retry with backoff
500	Server error	Retry with exponential backoff
503	Service unavailable	Wait and retry

7.2 Retry Logic

```
import time

def make_request_with_retry(self, endpoint: str, max_retries: int = 3) -> dict:
    """Make API request with retry logic."""

    for attempt in range(max_retries):
        try:
            return self.make_api_request('GET', endpoint)
        except requests.exceptions.Timeout:
            if attempt < max_retries - 1:
                time.sleep(5 * (attempt + 1))  # Exponential backoff
                continue
            raise
        except requests.exceptions.HTTPError as e:
            if e.response.status_code == 429:  # Rate limited
                time.sleep(30)
                continue
            raise
```

7.3 Async Query States

State	Meaning
`Waiting`	Query queued
`Processing`	Query executing
`Complete`	Data ready
`Failed`	Query failed (check `ErrorDetails`)

8. Complete Sync Example

```
#!/usr/bin/env python3
"""Complete Wild Apricot sync example."""

import json
import base64
import time
import sqlite3
import requests
from datetime import datetime, timezone

class WildApricotSync:
    def __init__(self, config_file: str):
        with open(config_file) as f:
            config = json.load(f)

        self.api_key = config['wildapricot']['api_key']
```

```
        self.account_id = config['wildapricot']['account_id']
        self.db_path = config.get('database_path', 'wa.db')
        self.token_cache = None
        self.token_expiry = None

    def get_access_token(self) -> str:
        if self.token_cache and self.token_expiry and time.time() < self.token_expiry:
            return self.token_cache

        credentials = base64.b64encode(f"APIKEY:{self.api_key}".encode()).decode()

        response = requests.post(
            "https://oauth.wildapricot.org/auth/token",
            headers={
                "Authorization": f"Basic {credentials}",
                "Content-Type": "application/x-www-form-urlencoded"
            },
            data={"grant_type": "client_credentials", "scope": "auto"}
        )
        response.raise_for_status()

        data = response.json()
        self.token_cache = data['access_token']
        self.token_expiry = time.time() + data.get('expires_in', 1800) - 300

        return self.token_cache

    def api_get(self, endpoint: str, params: dict = None) -> dict:
        url = f"https://api.wildapricot.org/v2.2/accounts/{self.account_id}/{endpoint}"

        response = requests.get(
            url,
            headers={
                'Authorization': f'Bearer {self.get_access_token()}',
                'Accept': 'application/json'
            },
            params=params,
            timeout=30
        )
        response.raise_for_status()
        return response.json()

    def poll_async(self, result_url: str) -> list:
        for _ in range(40):
            time.sleep(3)
            response = requests.get(
                result_url,
                headers={'Authorization': f'Bearer {self.get_access_token()}'},
                timeout=30
            )
            result = response.json()

            if result.get('State') == 'Complete':
```

```
                return result.get('Contacts', [])
            elif result.get('State') == 'Failed':
                raise Exception(result.get('ErrorDetails'))

        raise TimeoutError("Async query timeout")

    def fetch_contacts(self) -> list:
        response = self.api_get('contacts', {'$async': 'true'})
        return self.poll_async(response['ResultUrl'])

    def fetch_events(self) -> list:
        all_events = []
        skip = 0

        while True:
            response = self.api_get('events', {'$top': 100, '$skip': skip})
            events = response.get('Events', [])

            if not events:
                break

            all_events.extend(events)

            if len(events) < 100:
                break
            skip += 100

        return all_events

    def sync(self):
        print("Fetching contacts...")
        contacts = self.fetch_contacts()
        print(f"  Retrieved {len(contacts)} contacts")

        print("Fetching events...")
        events = self.fetch_events()
        print(f"  Retrieved {len(events)} events")

        # Save to database...
        print("Sync complete!")


if __name__ == "__main__":
    sync = WildApricotSync('wa_config.json')
    sync.sync()
```

9. API Rate Limits

Wild Apricot enforces rate limits:

- **General:** ~100 requests per minute

- **Async queries:** Limited concurrent queries
- **Best practice:** Add delays between bulk operations

---

10. Troubleshooting

10.1 Check Token

```
curl -X POST "https://oauth.wildapricot.org/auth/token" \
  -H "Authorization: Basic $(echo -n 'APIKEY:YOUR_API_KEY' | base64)" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=client_credentials&scope=auto"
```

10.2 Test API Call

```
TOKEN="your_access_token"
curl -s "https://api.wildapricot.org/v2.2/accounts/176353/membershiplevels" \
  -H "Authorization: Bearer $TOKEN" | python3 -m json.tool
```

10.3 Check Sync Logs

```
# On server
tail -100 /home/sbnewcom/wa_sync/logs/wa_sync.log
```

---

11. References

- Wild Apricot API Documentation
- Wild Apricot Authentication Guide
- OData Query Options