

Achievements Classic
Dokumentation der öffentlichen API

Addon-Version 1.00 | Public API 1.4.0
Classic Era und TBC Classic

Diese Dokumentation wurde direkt aus dem aktuellen Quellstand `api/PublicAPI.lua` und den zugehörigen API-Modulen erzeugt. Methoden mit führendem Unterstrich sind interne Integrationsflächen und gehören ausdrücklich nicht zum öffentlichen Vertrag.

PDF-Navigation: Die Kapitel erscheinen als Lesezeichen/Gliederung im linken Navigationsbereich des PDF-Readers. Zusätzlich folgt ein klickbares Inhaltsverzeichnis.

Inhaltsverzeichnis

1. Überblick 4

2. Schnellstart 4

2.1 TOC des Extension-Addons . . . 4
2.2 Modul registrieren . . . 4
2.3 Ersten Erfolg registrieren . . . 5

3. Modul, Speicher und Registrierung 5

3.1 RegisterModule . . . 5
3.2 Speicherlayout und Scope . . . 5
3.3 Asynchrone Registrierung . . . 5

4. Kriterien und Trigger 6

4.1 Eigene Kriterien-Typen . . . 6
4.2 Trigger-Modi . . . 6
4.3 ODER-Kriterien . . . 6

5. Kategorien und Erfolge 6

5.1 RegisterCategory . . . 6
5.2 RegisterAchievement . . . 7
Kriterien-Definition . . . 7
Erfolgsketten und Meta-Erfolge . . . 7

6. Titelbelohnungen 7

7. Fortschrittsleisten und UI-Tabs 8

7.1 Summary-Fortschrittsleiste . . . 8
7.2 Leerer Extension-Tab . . . 8
7.3 Nativer Erfolgstab . . . 8
7.4 Kategorien in leeren Tabs . . . 8
7.5 Fortschrittsleisten innerhalb eines Tabs . . . 8

8. Events und Aktualisierung 8

9. Handles, Fehler und Stabilität 9

9.1 Read-only Handles . . . 9
9.2 Validierungsfehler . . . 9
9.3 Geschützte Callbacks . . . 9
9.4 Kompatibilitätsregel . . . 9

10. Vollständiges Beispiel-Addon 9

Konstanten und Grenzen 10

Öffentliche ID-Bereiche . . . 11

Referenz: globales API-Objekt 11

Snapshot-Strukturen 11

Referenz: Modul-Handle **12**

Anhang: Kriterien-Typen **12**

Schluss / Support-Hinweis **16**

1. Überblick

Die öffentliche API von Achievements Classic ist eine Erweiterungsschnittstelle für andere WoW-Addons. Sie erlaubt eigene Kategorien, Erfolge, Kriterien, Fortschrittsleisten und UI-Tabs, ohne die internen Datenstrukturen von Achievements Classic direkt zu verändern.

- Globaler Einstiegspunkt: `_G.AchievementsClassicAPI`
- Aktuelle API-Version: 1.4.0
- Aktuelle Addon-Version: 1.00
- Unterstützte Clients: Classic Era und TBC Classic/Anniversary
- Extension-Fortschritt wird ausschließlich in einer vom Extension-Addon bereitgestellten SavedVariables-Tabelle gespeichert.

Nur Methoden und Felder ohne führenden Unterstrich sind Teil der öffentlichen API. Methoden wie `_GetTabs()` oder `_TriggerExtensions()` sind interne Integrationsflächen und dürfen von Extensions nicht verwendet werden.

2. Schnellstart

2.1 TOC des Extension-Addons

Das Extension-Addon sollte Achievements Classic als Abhängigkeit laden und eine eigene SavedVariables-Tabelle deklarieren. Die Persistenz kann accountweit oder pro Charakter gewählt werden.

```
## Dependencies: AchievementsClassic
## SavedVariables: MyAchievementsExtensionDB

Core.lua
```

2.2 Modul registrieren

```
MyAchievementsExtensionDB = MyAchievementsExtensionDB or {}

local API = _G.AchievementsClassicAPI
assert(API, "Achievements Classic is required")
assert(API:IsCompatible(1, 4), "Achievements Classic API 1.4+ required")

local mod = API:RegisterModule("my_extension", {
    name = "My Achievement Pack",
    version = "1.0.0",
    author = "Example Author",
    storage = MyAchievementsExtensionDB,
})
```

`metadata.storage` ist nicht optional. Die Tabelle muss dem Extension-Addon gehören. AC_LocalData, AC_AccountData, AC_Settings, AC_CharacterPoints oder darin verschachtelte Tabellen werden ausdrücklich abgelehnt.

2.3 Ersten Erfolg registrieren

```
local category = mod:RegisterCategory("demo", {
    name = "Demo",
    tab = API.TAB.PLAYER,
    order = 900,
})

local eventType = mod:RegisterCriteriaType("DEMO_EVENT", 1)

local achievement = mod:RegisterAchievement("first_demo", {
    category = category,
    name = "First Demo",
    description = "Trigger the demo event ten times.",
    points = 10,
    icon = "Interface\\Icons\\INV_Misc_Note_01",
    criteria = {
        {
            key = "count",
            type = eventType,
            data = { "sample" },
            quantity = 10,
        },
    },
})

-- Later, when the extension's real game event happens:
mod:Trigger(eventType, { "sample" }, 1, "increment")
```

3. Modul, Speicher und Registrierung

3.1 RegisterModule

`API:RegisterModule(moduleID, metadata)` erzeugt das Modul-Handle. `moduleID` muss ein nicht leerer Schlüssel aus Buchstaben, Zahlen, Unterstrich, Punkt oder Bindestrich sein. Eine bereits registrierte Modul-ID liefert dasselbe Handle zurück, solange dieselbe Speichertabelle verwendet wird.

metadata-Feld	Pflicht	Bedeutung
storage	ja	Eigene SavedVariables-Tabelle der Extension
name	nein	Anzeigename; Standard = moduleID
version	nein	Versionsstring; Standard = "unknown"
author	nein	Autorname

3.2 Speicherlayout und Scope

Achievements Classic legt unterhalb der bereitgestellten Tabelle einen verwalteten Bereich `storage.AchievementsClassic` mit Schema-Version 1 und einem `modules`-Teilbaum an. Dieser Teilbaum sollte von der Extension nicht direkt verändert werden.

Ob Fortschritt accountweit oder pro Charakter gilt, entscheidet die Extension durch `## SavedVariables` bzw. `## SavedVariablesPerCharacter` in ihrer eigenen TOC.

3.3 Asynchrone Registrierung

Registrierung wird in kleinen Slices abgeschlossen, damit große Erweiterungspakete Login/Reload nicht blockieren. Pro Slice werden höchstens 20 Abschlussprüfungen und ungefähr 2 ms Profilzeit verwendet.

- `BeginBulkRegistration()` hält die Verarbeitung zurück und kann verschachtelt werden.
- `EndBulkRegistration()` beendet eine Ebene und startet die Verarbeitung, sobald die Tiefe 0 erreicht.
- `IsRegistrationPending()` zeigt an, ob Bulk-Registrierung, Struktur-Commit oder Abschlussprüfungen noch offen sind.
- `REGISTRATION_COMMITTED` meldet den endgültigen Commit eines Moduls.

```
mod:BeginBulkRegistration()
-- Register many categories/achievements/tabs here.
mod:EndBulkRegistration()

local token = mod:Subscribe(API.EVENT.REGISTRATION_COMMITTED,
```

```
    function(module, committedModuleID, changeCount)
        if committedModuleID == module:GetID() then
            print("Registration committed:", changeCount)
        end
    end)
```

4. Kriterien und Trigger

4.1 Eigene Kriterien-Typen

`RegisterCriteriaType(key, dataLength, options)` registriert einen Extension-eigenen Typ. `dataLength` darf 0 bis 16 betragen. Optional kann `options.id` eine explizite ID aus dem öffentlichen Kriterien-Typ-Bereich enthalten.

```
local typeID = mod:RegisterCriteriaType("MY_COUNTER", 2)

-- A matching criterion might use:
-- type = typeID
-- data = { "zone", 42 }

mod:Trigger(typeID, { "zone", 42 }, 1, "increment")
```

4.2 Trigger-Modi

mode	Verhalten
nil / "increment"	count zum bisherigen Fortschritt addieren
"set"	Fortschritt auf count setzen; wird auf 0..quantity begrenzt

Kriterien ohne `quantity` werden beim ersten passenden Trigger direkt abgeschlossen. Kriterien mit Menge werden auf den Bereich 0 bis quantity begrenzt.

`module:Trigger()` nutzt denselben zentralen Progressionspfad wie das Hauptaddon. Wird ein Core-Kriterientyp getriggert, können dadurch auch passende Kernkriterien fortschreiten. Für rein Extension-eigene Ereignisse daher bevorzugt einen eigenen Kriterien-Typ registrieren.

4.3 ODER-Kriterien

Der interne Kriterien-Typ 0R ist über die Public API gesperrt. Stattdessen am Erfolg `anyCriteria = true` setzen.

5. Kategorien und Erfolge

5.1 RegisterCategory

Feld	Typ	Beschreibung
name	string	Pflicht; lokalisierter Kategoriename
tab	Tab/Key/ID	Optional; Standard = Spieler-Tab
parent	Handle/Key/ID	Optional; muss im selben Tab liegen
id	integer	Optionale explizite Public-ID
order	number	Sortierwert; Standard 100
unavailable	boolean	Kategorie als nicht verfügbar markieren
featsOfStrength	boolean	Nur in API-Erfolgstabs; nur Hauptkategorie; maximal eine je Tab

Extensions dürfen keine Kategorie oder Erfolge unterhalb der nativen Kernkategorie „Heldentaten“ registrieren. Eigene Heldentaten-Kategorien sind nur innerhalb eines eigenen API-Erfolgstabs erlaubt.

5.2 RegisterAchievement

Feld	Pflicht	Bedeutung
category	ja	Kategorie-Handle, Modul-Key oder numerische Kategorie-ID
name	ja	Name
description	ja	Beschreibung; leerer String erlaubt
points	nein	Nicht negative Zahl; Standard 0
icon	ja	Texturpfad, Iconname oder File-ID
criteria	ja	Nicht leeres Array von Kriterien
id	nein	Explizite Public-ID
hideCriteriaUI	nein	Alle Kriterien in der UI ausblenden
anyCriteria	nein	Ein beliebiges Kriterium reicht für Abschluss
previous	nein	Vorgänger aus demselben Extension-Modul
faction	nein	"Horde" oder "Alliance"
rewardText	nein	Belohnungstext
titleReward	nein	Titeldefinition
unavailable	nein	Nicht verfügbar
priority	nein	Numerische Priorität

Kriterien-Definition

Feld	Pflicht	Bedeutung
key	ja	Modulinterner stabiler Schlüssel
type	ja	Core-Typname/ID oder eigener Typ-Key/ID
data	abhängig	Adressdaten; Anzahl muss zur Typ-Arity passen
quantity	nein	Zielmenge > 0
name	nein	Kriterientext
id	nein	Explizite Public-ID
hidden	nein	Dieses Kriterium in UI ausblenden
quantityFormatter	nein	function(current, required) -> string

Erfolgsketten und Meta-Erfolge

`previous` darf nur auf einen Erfolg desselben Extension-Moduls zeigen. Wird ein späterer Erfolg der Kette abgeschlossen, werden noch offene Vorgänger desselben Moduls rekursiv abgeschlossen. Für Meta-Erfolge kann `COMPLETE_ACHIEVEMENT` verwendet werden; abhängige Meta-Erfolge werden beim Abschluss des referenzierten Erfolgs gezielt neu geprüft.

6. Titelbelohnungen

Ein Erfolg mit `titleReward` benötigt zusätzlich einen nicht leeren `rewardText`. `titleReward = true` verwendet Standardwerte; alternativ kann eine Tabelle angegeben werden.

Feld	Bedeutung
key	Optionaler stabiler Titel-Key
format	Formatstring für männlich/Standard
femaleFormat	Optionaler weiblicher Formatstring
nativeMask	Optionaler positiver Blizzard-Titelmaskenwert
nativeMaskFemale	Optionaler weiblicher Maskenwert

```
rewardText = "Title: the Explorer",
titleReward = {
```

```
    key = "EXPLORER",
    format = "%s the Explorer",
    femaleFormat = "%s the Explorer",
}
```

7. Fortschrittsleisten und UI-Tabs

7.1 Summary-Fortschrittsleiste

`RegisterSummaryProgressBar()` fügt der Zusammenfassung eine Extension-Leiste hinzu. Insgesamt sind höchstens vier öffentliche Extension-Summary-Bars registrierbar.

Feld	Bedeutung
label	Pflicht
category	Optional; rekursiver Fortschritt wird automatisch aus verfügbaren Erfolgen gezählt
getProgress	function(module, context) -> current, maximum, text; erforderlich wenn keine category
isVisible	function(module, context) -> boolean
tooltip	string oder function(module, context) -> value
onClick	function(module, button)
order	Sortierwert; Standard 100

7.2 Leerer Extension-Tab

`RegisterTab()` erzeugt einen frei zeichnenden Tab. Definition: `name` (Pflicht), optional `title, icon, tooltip, width, order, createFrame, summaryName, onShow, onHide, onRefresh`. Die Breite liegt zwischen 80 und 160 Pixeln, Standard 115.

Callback-Signaturen: `createFrame(module, parent)`; `onShow/onHide/onRefresh(module, frame)`; Tooltip-Funktion `tooltip(module)`.

7.3 Nativer Erfolgstab

`RegisterAchievementTab()` erzeugt einen eigenen Erfolgstab, der die native Achievement-UI verwendet. `createFrame` ist hier nicht erlaubt. Kategorien werden über `RegisterCategory()` mit diesem Tab-Handle angelegt. Maximal 10 Hauptkategorien je API-Erfolgstab.

7.4 Kategorien in leeren Tabs

`RegisterTabCategory()` ist nur für leere Tabs verfügbar. `name` und `createFrame` sind Pflicht; `parent, order, onShow, onHide, onRefresh, tooltip` sind optional.

7.5 Fortschrittsleisten innerhalb eines Tabs

`RegisterTabProgressBar()` erlaubt höchstens 10 Leisten pro Tab und erzeugt jeweils eine Hauptkategorie als Navigationsziel. In Erfolgstabs wird Fortschritt automatisch aus der Kategorie berechnet; `getProgress` und `createFrame` sind dort nicht erlaubt. In leeren Tabs sind beide erforderlich.

8. Events und Aktualisierung

`module:Subscribe(eventName, callback)` registriert einen Listener. Der Callback erhält das Modul-Handle automatisch als erstes Argument. `Unsubscribe(token)` entfernt ihn.

Event	Callback-Argumente nach module	Bedeutung
PROGRESS_CHANGED	criteriaType, data	Fortschritt wurde verändert
ACHIEVEMENT_COMPLETED	achievementID, sourceModuleID?	Erfolg abgeschlossen; sourceModuleID ist bei Extension-Abschlüssen vorhanden
TITLE_CHANGED	selectedTitle	Aktiver Addon-Titel geändert

Event	Callback-Argumente nach module	Bedeutung
DATABASE_CHANGED	kind, objectID, moduleID, changes, phase	Registrierungs-/Datenbankänderung; Extension-Batches verwenden kind="batch"
REGISTRATION_COMMITTED	moduleID, changeCount	Registrierung eines Moduls vollständig committed

```
local token = mod:Subscribe(API.EVENT.ACHIEVEMENT_COMPLETED,
    function(module, achievementID, sourceModuleID)
        print("Completed:", achievementID, sourceModuleID)
    end)

-- Later:
mod:Unsubscribe(token)
```

`RequestUIRefresh()` sendet eine gezielte Extension-UI-Aktualisierung für das aufrufende Modul. Es ist für eigene UI-Zustände gedacht, die sich geändert haben, ohne dass eine neue Registrierung nötig ist.

9. Handles, Fehler und Stabilität

9.1 Read-only Handles

Registrierungs-Handles sind schreibgeschützt. Das Setzen eigener Felder löst einen API-Fehler aus. Kategorie-Handles enthalten u. a. `id, key, moduleID, tabID, kind`; Erfolgs-Handles zusätzlich `blizzardID, categoryID, criteriaIDs`. Tab-Handles enthalten `key, moduleID, kind, tabType, tabID`.

9.2 Validierungsfehler

Fehlerhafte Registrierungen lösen Lua-Fehler mit Präfix `AchievementsClassicAPI:` aus. Das betrifft z. B. ungültige Keys, falsche Typen, ID-Kollisionen, verbotene Feats-of-Strength-Ziele oder fehlende Pflichtfelder.

9.3 Geschützte Callbacks

UI-Callbacks von Extensions werden intern über `pcall` ausgeführt. Fehler werden mit Modul-ID und Callback-Oberfläche an den WoW-Fehlerhandler gemeldet, statt die gesamte UI-Ausführung ungefangen abzubrechen.

9.4 Kompatibilitätsregel

Vor API-Nutzung sollte mindestens `API:IsCompatible(1, 4)` geprüft werden. Ein Major-Wechsel ist inkompatibel; höhere Minor-Versionen innerhalb desselben Majors sind kompatibel, wenn die benötigte Mindest-Minor-Version erfüllt ist.

10. Vollständiges Beispiel-Addon

Das folgende kompakte Beispiel zeigt eigene SavedVariables, Bulk-Registrierung, einen eigenen Erfolgstab, eine Kategorie, einen eigenen Trigger-Typ und ein registriertes Event.

```
MyPackDB = MyPackDB or {}
local API = assert(_G.AchievementsClassicAPI)
assert(API:IsCompatible(1, 4))

local mod = API:RegisterModule("example_pack", {
    name = "Example Pack",
    version = "1.0.0",
    storage = MyPackDB,
})

mod:BeginBulkRegistration()

local tab = mod:RegisterAchievementTab("pack_tab", {
    name = "Example",
    title = "Example Achievements",
    width = 115,
    order = 500,
})

local category = mod:RegisterCategory("general", {
    name = "General",
```

```
    tab = tab,
    order = 10,
})

local typeID = mod:RegisterCriteriaType("TOKEN_COLLECTED", 1)

mod:RegisterAchievement("collector", {
    category = category,
    name = "Collector",
    description = "Collect 25 example tokens.",
    points = 10,
    icon = "Interface\\Icons\\INV_Misc_Coin_01",
    criteria = {
        {
            key = "tokens",
            name = "Example tokens",
            type = typeID,
            data = { "default" },
            quantity = 25,
        },
    },
})

mod:RegisterTabProgressBar(tab, "general_progress", {
    name = "General",
    label = "General progress",
    order = 10,
})

mod:EndBulkRegistration()

-- Call from your real game-event handler:
local function OnTokenCollected(amount)
    mod:Trigger(typeID, { "default" }, amount or 1, "increment")
end

mod:Subscribe(API.EVENT.REGISTRATION_COMMITTED,
    function(module, moduleID, changeCount)
        if moduleID == module:GetID() then
            print("Example Pack ready; changes:", changeCount)
        end
    end)
```

Eigene Texte sollten im Extension-Addon selbst lokalisiert werden. Die Public API übernimmt die übergebenen Strings unverändert und erzwingt keine bestimmte Locale-Struktur.

Konstanten und Grenzen

Das globale API-Objekt enthält stabile Konstanten für Tabs, Events, Clienttypen, Limits, Speicherregeln, ID-Bereiche und alle registrierten Kriterien-Typen.

Konstante	Wert / Bedeutung
API.API_VERSION	"1.4.0"
API.TAB.PLAYER	Spieler-/Erfolgs-Tab
API.TAB.GUILD	Gilden-Tab
API.TAB.STATS	Statistik-Tab
API.TAB_TYPE.EMPTY	"empty"
API.TAB_TYPE.ACHIEVEMENTS	"achievements"
API.LIMITS.SUMMARY_PROGRESS_BARS	4 (global für Extension-Fortschrittsleisten)
API.LIMITS.TAB_PROGRESS_BARS	10 je Tab
API.LIMITS.ACHIEVEMENT_TAB_MAIN_CATEGORIES	10 Hauptkategorien je API-Erfolgstab
API.LIMITS.REGISTRATION_CHECKS_PER_SLICE	20 Abschlussprüfungen pro Registrierungs-Slice
API.LIMITS.REGISTRATION_TIME_BUDGET_MS	2 ms Zeitbudget pro Registrierungs-Slice
API.STORAGE.SCHEMA_VERSION	1

Konstante	Wert / Bedeutung
API.STORAGE.ROOT_KEY	"AchievementsClassic"
API.STORAGE.POLICY	"extension-owned-only"

Öffentliche ID-Bereiche

Art	Erste ID	Letzte ID
Category / Kategorie	1,000,000	99,999,999
Achievement / Erfolg	100,000,000	999,999,999
Criteria / Kriterium	1,000,000,000	1,999,999,999
Criteria type / Kriterientyp	10,000	99,999

IDs werden standardmäßig deterministisch aus `moduleID + kind + key` erzeugt. Solange Modul-ID und Schlüssel stabil bleiben, bleibt auch die berechnete ID stabil. Bei einer Kollision muss eine explizite ID aus dem jeweiligen öffentlichen Bereich vergeben oder der Schlüssel geändert werden.

Referenz: globales API-Objekt

Methode	Rückgabe	Beschreibung
GetVersion()	major, minor, patch	Liefert 1, 4, 0.
GetAddonVersion()	string	Liest die Addon-Version aus der TOC (aktuell 1.00).
GetClientFlavor()	string	CLASSIC_ERA oder TBC_ANNIVERSARY.
GetClientInterfaceVersion()	number	Aktuelle Blizzard-Interface-ID des Clients.
IsClassicEra()	boolean	Classic-Era-Erkennung.
IsTBCAnniversary()	boolean	TBC-Client-Erkennung.
IsCompatible(requiredMajor, minimumMinor)	boolean	Major muss exakt passen; API-Minor muss mindestens minimumMinor sein.
RegisterModule(moduleID, metadata)	module handle	Registriert oder liefert das Modul. metadata.storage ist erforderlich.
GetModule(moduleID)	module handle | nil	Liefert ein bereits registriertes Modul.
GetRegisteredModules()	array	Snapshots mit id, name, version, author.
GetCategory(categoryID)	table | nil	Read-only Snapshot einer Kategorie.
GetAchievement(achievementID)	table | nil	Read-only Snapshot eines Erfolgs samt Kriterien-Metadaten.
GetEarnedTitles()	array	Verdiente Addon-Titel als Snapshots.
GetSelectedTitle()	table | nil	Aktuell ausgewählter Addon-Titel.
IsAchievementCompleted(achievementID)	boolean	Lokaler Abschlussstatus für Kern- oder Extension-Erfolg.
GetCriteriaProgress(achievementID, criteriaID)	number	Aktueller lokaler Kriterienfortschritt.
GetRegisteredSummaryProgressBars()	array	Öffentliche Übersicht der registrierten Summary-Bars.
GetRegisteredTabs()	array	Öffentliche Übersicht der registrierten Extension-Tabs.
IsExtensionAchievement(achievementID)	boolean	True, wenn der Erfolg durch die Public API registriert wurde.

Snapshot-Strukturen

`GetCategory()` liefert: `id, name, parentID, tabID, order, featsOfStrength, available`.

`GetAchievement()` liefert: `id, blizzardID, name, description, points, icon, rewardText, titleReward, categoryID, tabID, available, hideCriteriaUI, criteria`. Jeder Kriterien-Snapshot enthält `id, name, type, quantity, hidden`.

Titel-Snapshots enthalten `key, label, format, femaleFormat, nativeMask, nativeMaskFemale, achievementID, sourceModuleID`.

Referenz: Modul-Handle

Methode	Rückgabe
GetID()	string
GetMetadata()	table
BeginBulkRegistration()	self
EndBulkRegistration()	self
IsRegistrationPending()	boolean
RegisterCriteriaType(key, dataLength, options)	criteria type ID
RegisterCategory(key, definition)	category handle
RegisterAchievement(key, definition)	achievement handle
RegisterSummaryProgressBar(key, definition)	read-only handle
RegisterTab(key, definition)	tab handle
RegisterAchievementTab(key, definition)	tab handle
RegisterTabCategory(tab, key, definition)	tab-category handle
RegisterTabProgressBar(tab, key, definition)	category handle
RequestUIRefresh()	nil
Trigger(criteriaType, data, count, mode)	boolean changed
Subscribe(eventName, callback)	subscription token
Unsubscribe(token)	boolean
GetCategoryID(key)	number | nil
GetAchievementID(key)	number | nil
GetCriteriaID(achievementKey, criteriaKey)	number | nil
GetCriteriaTypeID(key)	number | nil
HasSummaryProgressBar(key)	boolean
HasTab(key)	boolean
GetTabHandle(key)	handle | nil
HasTabProgressBar(tab, key)	boolean
IsAchievementCompleted(value)	boolean
GetCriteriaProgress(achievement, criteria)	number
ResetProgress()	nil

`ResetProgress()` löscht den gesamten gespeicherten Fortschritt dieses Extension-Moduls und initialisiert dessen registrierte Erfolge/Kriterien neu. Es berührt keine anderen Module und keine Kernfortschritte, ist aber innerhalb des eigenen Moduls destruktiv.

Anhang: Kriterien-Typen

`API.CRITERIA_TYPE` enthält die folgenden stabilen Typnamen und IDs. Die Spalte „Datenwerte“ ist die Anzahl der Werte, die in `data` zur Adressierung des Triggers verwendet werden. Die genaue fachliche Bedeutung der Datenwerte ist typabhängig und wird von der jeweiligen Kernlogik bestimmt.

Für eigene Extension-Ereignisse ist `module:RegisterCriteriaType()` die sicherste Wahl. Das manuelle Triggern eines Core-Typs kann auch passende Kernkriterien von Achievements Classic fortschreiben. `OR` ist

ausdrücklich nicht öffentlich nutzbar; für ODER-Erfolge any`Criteria = true` verwenden.

ID	API.CRITERIA_TYPE.*	Datenwerte
1	KILL_NPC	1
2	KILL_NPCS	1
3	COMPLETE_QUEST	1
4	COMPLETE_QUESTS	0
5	OBTAIN_ITEM	1
6	KILL_PLAYERS	0
7	KILL_PLAYER_OF_RACE	1
8	KILL_PLAYER_OF_CLASS	1
9	ATIESH	0
10	REACH_PVP_RANK	1
11	NOT_WORKING	0
12	REACH_PROFESSION_LEVEL	2
13	REACH_MAIN_PROFESSION_LEVEL	1
14	REACH_SECONDARY_PROFESSION_LEVEL	1
15	CRAFT_ITEM	1
16	REACH_REPUTATION	2
17	REACH_ANY_REPUTATION	1
18	EXPLORE_AREA	1
19	REACH_LEVEL	1
20	COMPLETE_ACHIEVEMENT	1
21	BANK_SLOTS	0
22	LOOT_QUEST_GOLD	0
23	BATTLEFIELD_STAT_MAX	2
24	BATTLEFIELD_SCORE_MAX	2
25	BATTLEFIELDS_SCORE	1
26	BATTLEFIELD_WINS	1
27	GEAR_QUALITY	2
28	FISH_AN_ITEM	1
29	LEARN_PROFESSION_RECIPE	2
30	LEARN_PROFESSION_RECIPES	1
31	DUELS	0
32	BATTLEFIELD_MAX_LEVEL_PARTICIPATION	0
33	EMOTE	2
34	BATTLEFIELD_FAST_WIN	1
35	BATTLEFIELD_STAT	2
36	SPECIAL	1
37	BOSS_WITHOUT_MOBS	1
38	BOSS_WITH_ALL_ALIVE	1
39	OR (internal OR)	0
40	KILL_NPC_HEROIC	1
41	ALTERAC_VALLEY_MINE_CAPTURE_MAX	0
42	BG_EYE_GLORY	0
43	BG_EYE_BERSERK	0

ID	API.CRITERIA_TYPE.*	Datenwerte
44	BG_POINTS	3
45	P3_FIRST_WEEK	1
46	FISH_ANY_ITEM	1
47	HAS_BUFF	1
48	KILL_ANY_NPC	0
49	COMPLETE_DAILY_QUEST	1
50	COMPLETE_DAILY_QUESTS	0
51	ARENA_WIN	0
52	ARENA_MAP	1
53	ARENA_RATING	1
54	ARENA_5V5_SURVIVOR	0
55	ARENA_HOT_STREAK	0
56	ARENA_HOTTER_STREAK	0
57	FISH_FROM_POOL	1
58	FISH_UP_BOSS	1
59	COMPANION_PETS	0
60	LOOT_MONEY	0
61	MOUNTS	0
62	READ_BOOK	1
63	DRINKS	0
64	FOODS	0
65	CLASSIC_FOODS	0
66	TBC_FOODS	0
67	TABARDS	0
68	SURVIVE_LONG_FALL	0
69	ALTERAC_BOTH_MINES_WIN	0
70	ALTERAC_KILL_MOUNTED_FIELD_OF_STRIFE	0
71	ALTERAC_VALLEY_ALL_STAR	1
72	ALTERAC_LOYAL_DEFENDER	1
73	ALTERAC_PERFECTION	1
74	ARATHI_MATCH_STAT	1
75	ARATHI_BASE_HONOR_KILLS	1
76	ARATHI_TERRITORIAL_WIN	0
77	ARATHI_RESILIENT_WIN	0
78	SPECIFIC_COMPANION_PET	2
79	OLD_SCHOOL_RIDE	0
80	SOUL_OF_IRON_LEVEL	0
81	SOUL_OF_IRON_BOSS	1
82	SOUL_OF_IRON_NAXX_IMMORTAL	0
83	HOME_ALONE	0
84	WARSONG_MATCH_STAT	1
85	WARSONG_MATCH_CONDITION	1
86	WARSONG_FLAG_CARRIER_KILLS	0
87	TIMED_ITEM_USES	1

ID	API.CRITERIA_TYPE.*	Datenwerte
88	CAST_SPELL	1
89	LOOT_ITEM	1
90	NOBLEGARDEN_BLUSHING_BRIDE	0
91	GNERD_HONORABLE_KILLS	0
92	WINTERVEIL_TIS_SEASON	0
93	WINTERVEIL_MISTLETOE_BROTHER	1
94	PVP_BG_WRECKING_BALL	0
95	PVP_BG_DAMAGE_CONTROL	0
96	PVP_HUG_DEAD_ENEMY	0
97	PVP_WAR_MOUNT	0
98	PVP_CITY_DEFENDER	0
99	PVP_BG_GRIM_REAPER	0
100	DUAL_TALENT_SPECIALIZATION	0
101	EYE_BASE_HONOR_KILLS	1
102	EYE_STORM_GLORY	0
103	EYE_FLAG_CARRIER_KILLS	0
104	EYE_DOMINATION_WIN	0
105	EYE_TARGET_BERSERKER_KILL	0
106	USE_ITEM	1
107	BREWFEST_DRUNKEN_STUPOR	0
108	KNOWN_SPELL	1
109	NATIVE_TITLE	1
110	SERVER_ONLY_FEAT	1
111	CHILDREN_WEEK_PVP_ACTION	1
112	CHILDREN_WEEK_DAILY_QUESTS	0
113	DAILY_QUEST_STREAK	0
114	MIDSUMMER_POLE_DANCE	0
115	WINTERVEIL_RACER_CRASHES	0
116	WINTERVEIL_OGRILA_REINDEER	0
117	LOOT_ROLL_100	2
118	OWN_ITEM	1
119	WEAPON_SKILLS_AT_LEVEL	1
120	TRIAGE_BANDAGE	1
121	TARGETED_ITEM_USE	3
122	KILL_PLAYER_IN_AREA	1
123	TIMED_SOLO_QUEST	2
124	EQUIP_ITEM	1
125	PVP_KILLING_BLOWS	0
126	ALTERAC_KILLING_BLOWS_TOTAL	0
127	PROFESSION_CRAFT	1
128	ALCHEMY_CRAFT	1
129	GATHER_PROFESSION	1
130	PROFESSION_SOURCE_ITEM	2
131	PROFESSION_SOURCE_ITEM_COUNT	2

ID	API.CRITERIA_TYPE.*	Datenwerte
132	PROFESSION_CRAFT_RECIPE_COUNT	2
133	PROFESSION_UNIQUE_OUTPUT_ITEMS	1
134	PROFESSION_CRAFT_OUTPUT	1
135	PROFESSION_CRAFT_QUALITY	2
136	PROFESSION_CRAFT_SLOT	2
137	ENCHANT_UNIQUE_SPELL_USED	0
138	ENCHANT_TARGET_SLOT	1
139	ENCHANT_TARGET_QUALITY	1
140	ENCHANT_TARGET_OWNER	1
141	ENCHANT_TARGET_OWNER_QUALITY	2
142	ENCHANT_TARGET_OWNER_SLOT	2
143	ENCHANT_TARGET_ITEM_LEVEL	1
144	DISENCHANT_TARGET_QUALITY	1
145	DISENCHANT_TARGET_SLOT	1
146	DISENCHANT_TARGET_ITEM_LEVEL	1

Schluss / Support-Hinweis

Diese Dokumentation beschreibt den öffentlichen Vertrag der API-Version 1.4.0 in Achievements Classic 1.00. Für die Kompatibilität einer Extension sind ausschließlich die dokumentierten öffentlichen Felder und Methoden maßgeblich. Interne Dateien und Methoden mit Unterstrich können sich ohne API-Versionsgarantie ändern.