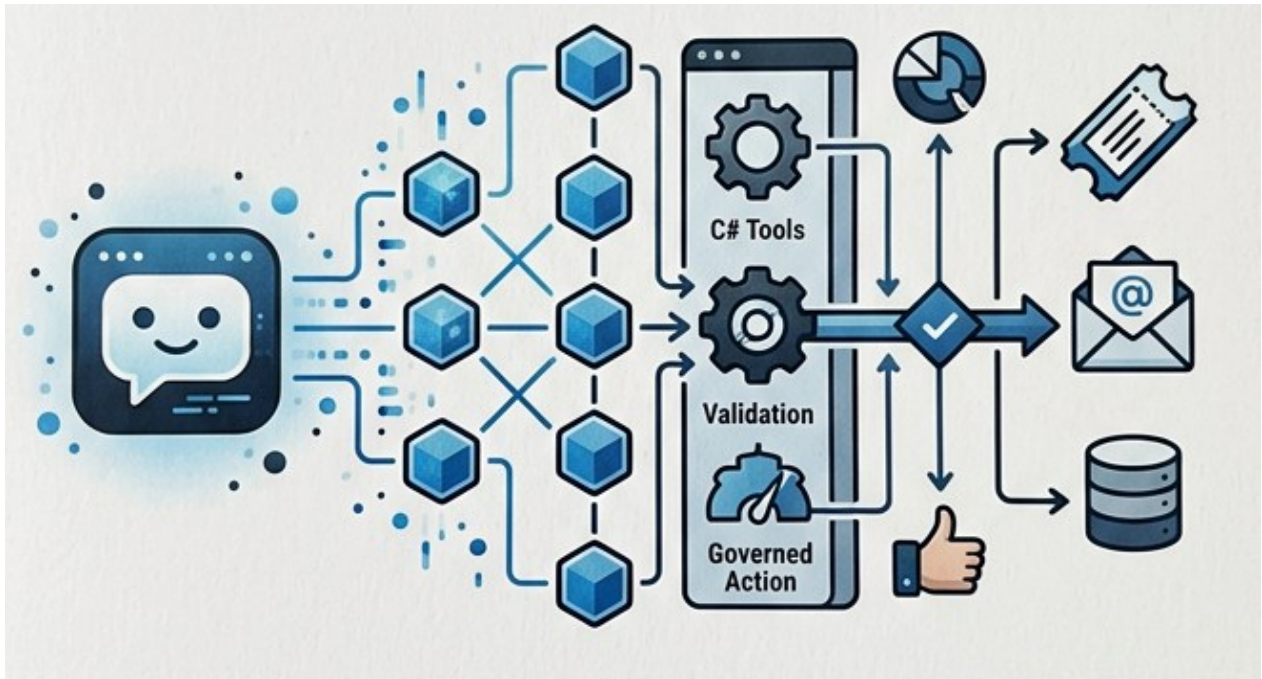


Cadrer un premier POC d'agent IA en .NET

Les 7 décisions à prendre avant de développer avec Microsoft Agent Framework



Un cas d'usage à préciser, des outils à délimiter et des résultats à vérifier avant de décider de la suite.

Olivier Dahan

Architecte et expert indépendant C# et .NET
Microsoft MVP depuis 2009

Édition octobre 2026

www.e-naxos.com
odahan@e-naxos.com

LE PÉRIMÈTRE DU GUIDE

Donner au POC une question à résoudre

Un agent produit une réponse convaincante, consulte une donnée et prépare une action. La démonstration montre une possibilité. Pour décider d'investir davantage, vous avez besoin de savoir si cette possibilité répond à votre problème, dans votre système et avec vos contraintes.

Un POC, ou preuve de concept, sert à examiner une hypothèse sur un périmètre limité. Sa réussite dépend de la qualité des éléments obtenus pour prendre une décision. Un résultat défavorable peut être utile s'il permet de comprendre pourquoi une approche doit être revue ou abandonnée.

Je m'adresse ici aux équipes .NET, architectes, éditeurs et responsables techniques qui souhaitent intégrer un agent à une application ou à un processus métier. Les sept décisions suivantes organisent le cadrage de cette expérimentation. Microsoft Agent Framework, abrégé MAF, constitue le support technique des ressources proposées en fin de document.

Situer le sujet

Nous parlons de construire un logiciel qui intègre l'IA. Organiser des agents qui écrivent le code de ce logiciel relève d'une autre question. Mon livre blanc « Développer avec les agents IA en gardant la maîtrise du projet » présente ce second sujet, avec le SDD et SAW. Les deux démarches peuvent se rencontrer dans un même projet.

Les sept décisions	Page
1 Définir le service attendu	3
2 Choisir le contexte d'exécution	4
3 Examiner la dépendance au fournisseur	5
4 Délimiter les outils accessibles	6
5 Organiser les autorisations et la validation	7
6 Définir un résultat exploitable	8
7 Préparer la décision de suite	9

Les pages 10 à 12 proposent une grille sans score, les sept erreurs à éviter et une fiche de cadrage appliquée au cas de support. Les pages 13 et 14 orientent vers les ressources MAF et l'accompagnement adapté. Vous pouvez utiliser ce guide sans avoir vu la vidéo d'initiation ni lu le code des exemples.

DÉCISION 01 / 07

Définir le service attendu

Un assistant conversationnel peut déjà rendre service : reformuler une demande, résumer un dossier ou retrouver une information. Un processus qui s'appuie sur un agent peut aussi sélectionner des outils et préparer des actions. Le choix doit partir du travail à améliorer et des contraintes du système.

Je vous conseille de décrire la situation actuelle avant de parler d'autonomie. Qui fait le travail ? Quelles informations consulte cette personne ? Quelle difficulté justifie une expérimentation ? Une réponse utile peut prendre la forme d'un brouillon, d'une proposition structurée ou d'une action soumise à autorisation.

Une hypothèse à examiner

Prenons un exemple fictif qui servira de fil conducteur : une application .NET reçoit des demandes de support. Un agent pourrait consulter les informations accessibles, préparer les champs d'un ticket et demander confirmation avant sa création. L'hypothèse porte sur l'aide à la préparation et sur la fiabilité du passage vers le système de tickets.

L'équipe compare cette approche à sa pratique actuelle, voire à une solution plus simple fondée sur un formulaire ou des règles. Elle examine l'effort de préparation, les corrections nécessaires et les erreurs introduites. Le recours à un agent doit être justifié par ce qu'il apporte dans ce contexte.

Choisir ce qui reste hors périmètre

Dans ce premier travail, l'agent ne ferme pas de ticket, ne modifie pas les droits et n'envoie pas de réponse au client. La création effective n'est accessible qu'après autorisation. Ces exclusions permettent de savoir ce que les résultats pourront éclairer et ce qu'ils laisseront à étudier.

Une capacité supplémentaire augmente le travail d'intégration et de vérification. Je commencerais avec le plus petit périmètre qui permet d'examiner l'hypothèse utile. Le nombre d'outils ou d'agents n'est pas, en lui-même, un résultat à rechercher.

À consigner. Le travail concerné, son utilisateur, le résultat attendu, l'hypothèse examinée, le point de comparaison et les exclusions.

Question de contrôle. Si le POC s'arrête demain, saurez-vous nommer la décision qu'il devait éclairer et les éléments qui vous manqueraient encore ?

Choisir le contexte d'exécution

Le choix entre local, cloud et hybride dépend des données, du matériel, du niveau de service et des capacités d'exploitation. Le POC doit permettre d'examiner les conséquences du choix retenu sans laisser croire que ses conditions seront celles de la production.

Une exécution locale peut être intéressante pour expérimenter avec certains modèles et limiter certains échanges de données. Elle ne garantit pas, à elle seule, la confidentialité. Les journaux, la télémétrie, les outils appelés et les autres services utilisés doivent également être examinés. Les coûts incluent le matériel, sa maintenance et le temps de l'équipe.

Une exécution distante peut donner accès à d'autres capacités et à des services d'exploitation. Il reste à examiner les conditions de traitement des données, les accès, les limites de service et la dépendance réseau. Une architecture hybride demande de préciser quels éléments s'exécutent où et quelles informations circulent entre eux.

Définir un environnement d'essai maîtrisé

Dans notre cas de support, l'équipe commence avec des demandes fictives ou des données préparées et autorisées pour cet usage. Elle identifie ce qui sera envoyé au modèle, ce qui sera lu dans l'application et ce qui pourra apparaître dans les traces. Un environnement de tickets dédié permet de tester la création sans affecter les utilisateurs réels.

Le jeu d'essai doit rester assez représentatif pour répondre à la question posée. Si l'anonymisation retire les difficultés qui font l'intérêt du cas, cette limite doit figurer dans le bilan. Le passage à d'autres données ou à un service réel demandera un examen supplémentaire.

Mesurer ce que vous utilisez réellement

Relevez le modèle, sa configuration, le matériel ou le service utilisé et les conditions d'exécution. Observez les temps d'attente, les échecs et la consommation pertinente pour votre contexte. Un bon résultat sur quelques demandes courtes ne caractérise pas un service soumis à d'autres volumes.

À consigner. Les flux de données, les accès, la localisation des composants, les conditions de mesure et les différences connues avec la cible envisagée.

Question de contrôle. Quelles conclusions deviendraient incertaines si vous changiez de matériel, de service ou de catégorie de données ?

Examiner la dépendance au fournisseur

Une séparation entre la logique métier, l'orchestration agentique et l'accès au modèle facilite l'évolution de l'application. Dans l'écosystème .NET, MAF peut s'appuyer sur des clients d'inférence de différents fournisseurs. Les fonctions disponibles dépendent cependant du fournisseur, de l'API, du type d'agent et du modèle retenus.

Je traiterais la réversibilité comme une propriété à éprouver. Une interface commune ne démontre pas que deux modèles appelleront les mêmes outils de la même façon, respecteront les mêmes formats ou produiront des résultats de qualité comparable. L'historique, l'authentification et les services hébergés peuvent aussi introduire des dépendances.

Séparer les contrats utiles

Le service métier doit conserver ses propres règles d'accès et de validation. L'orchestration prépare le contexte, expose les outils nécessaires et gère les échanges avec le modèle. L'adaptateur d'inférence porte les particularités de la connexion au fournisseur. Cette séparation permet d'identifier ce qu'un changement affecterait.

Dans l'exemple de support, la création d'un ticket reste une opération contrôlée par l'application. Changer de modèle ne doit pas lui transférer la responsabilité de vérifier les droits ou d'accepter les paramètres. Le nouveau modèle devra toutefois être évalué sur la préparation des champs et le choix des outils.

Décider ce qu'il faut comparer

Si la possibilité de changer de fournisseur est un objectif du POC, prévoyez une comparaison sur les mêmes cas et avec les mêmes critères. Conservez les configurations et les résultats. Sinon, indiquez que cette capacité reste à vérifier, même si l'architecture a été préparée pour faciliter une évolution.

Il n'est pas nécessaire de tester toutes les options pour commencer. Le besoin de comparaison dépend de la décision à prendre : disponibilité d'une fonction, contrainte de données, coût d'exploitation ou capacité de remplacement.

À consigner. Les responsabilités des composants, les fonctions dont vous dépendez, les options comparées et les limites de la réversibilité effectivement vérifiées.

Question de contrôle. Qu'est-ce qui changerait dans votre application si le fournisseur choisi ne pouvait plus être utilisé ?

Délimiter les outils accessibles

Les outils permettent à l'agent de consulter un état réel ou de demander une opération. MAF fournit des mécanismes pour exposer des fonctions à des agents compatibles. La conception de ces fonctions détermine une partie importante du périmètre d'action du système.

La distinction entre outils passifs et actifs reste utile pour commencer. Un outil de lecture consulte une information ; un outil d'écriture modifie un état ou déclenche un effet. La lecture mérite elle aussi des contrôles : une information peut être sensible, inappropriée pour cet utilisateur ou coûteuse à obtenir.

Exposer des opérations métier bornées

Dans notre exemple, l'agent peut consulter les catégories de support autorisées et préparer une proposition de ticket. La création utilise une opération spécifique qui vérifie ses paramètres et l'identité de l'utilisateur. Un accès générique à la base ou une fonction capable d'exécuter n'importe quelle requête élargirait fortement le risque.

Les descriptions des outils aident le modèle à comprendre leur usage. Elles ne remplacent pas les contrôles dans le code. La validation des arguments, les autorisations et les limites doivent être appliquées au point où l'opération s'exécute.

Préparer les échecs et les répétitions

Un appel peut échouer, prendre trop de temps ou produire une réponse ambiguë. L'équipe doit prévoir comment le système réagit et ce qui peut être réessayé. Une nouvelle tentative de création ne doit pas produire un ticket supplémentaire à l'insu de l'utilisateur. La détection des doublons et la stratégie de reprise font partie du périmètre à examiner.

Le contenu obtenu par un outil reste une donnée. Un texte de demande qui ordonne d'ignorer les règles ou d'envoyer des informations ailleurs ne devient pas une instruction autorisée. La réduction des permissions et les contrôles d'exécution restent nécessaires face aux erreurs et aux contenus hostiles.

À consigner. Pour chaque outil : l'opération, les données exposées, les droits, les effets possibles, les paramètres autorisés et la réaction à un échec ou une répétition.

Question de contrôle. L'opération resterait-elle encadrée si le modèle lui proposait des arguments plausibles mais inappropriés ?

Organiser les autorisations et la validation

Lorsqu'une opération a des conséquences, il faut décider qui peut la proposer, qui peut l'autoriser et dans quelles conditions elle s'exécute. Certaines opérations peuvent être préautorisées dans un périmètre limité ; d'autres demandent un accord explicite ou restent exclues de l'expérimentation.

MAF propose des mécanismes d'approbation d'outils pour les configurations concernées. L'application doit les intégrer à son parcours, présenter la demande à la bonne personne et traiter sa réponse. Ce mécanisme ne définit pas à votre place les rôles métier, les droits d'accès ou les informations nécessaires à la décision.

Montrer ce qui va être exécuté

Dans notre exemple, la personne examine le titre, la catégorie, le contenu et les destinataires du ticket avant sa création. Elle peut corriger ou refuser la proposition. Si les paramètres ou le contexte pertinent changent après l'accord, il faut déterminer si une nouvelle validation est nécessaire avant d'agir.

L'autorisation doit porter sur une opération identifiable. Elle ne doit pas devenir une permission générale pour les actions suivantes. Prévoyez aussi les cas d'absence de réponse, d'expiration, de refus et d'interruption du processus. Un ticket ne doit pas être créé parce qu'un parcours de validation s'est terminé de manière incertaine.

Conserver une trace qui permette de comprendre

La trace utile rapproche la proposition, les paramètres soumis, l'autorisation obtenue et le résultat de l'opération. Elle permet de distinguer une action proposée, une action autorisée et une action réellement réalisée. Le contenu journalisé doit rester compatible avec la sensibilité des données et les règles de conservation retenues.

Je vérifierais le parcours de refus aussi attentivement que le parcours d'accord. La présence d'un bouton de confirmation ne suffit pas : il faut établir que l'action attend effectivement l'autorisation requise et que le refus produit le comportement convenu.

À consigner. Les opérations libres, soumises à validation ou interdites, les personnes habilitées, les informations affichées et les traces nécessaires.

Question de contrôle. Pouvez-vous établir qu'une action exécutée correspond bien à ce que la personne avait autorisé ?

Définir un résultat exploitable

Une réponse bien rédigée peut être utile à un utilisateur. Une application peut avoir besoin d'un objet structuré, de champs validables ou d'une proposition d'action avec ses paramètres. Le format attendu doit être défini en fonction de ce que le système fera ensuite.

Les sorties structurées proposées dans certaines configurations MAF peuvent aider à obtenir un résultat conforme à un schéma. Leur disponibilité dépend notamment du modèle, du fournisseur et du type d'agent. Un objet conforme à un schéma peut encore contenir une information incorrecte ou une proposition métier inacceptable.

Distinguer les niveaux de vérification

Dans notre cas de support, une proposition contient un titre, une catégorie, une description et les références nécessaires. L'application vérifie la présence des champs, leurs types et leurs limites. Elle contrôle ensuite la validité de la catégorie et les règles propres au processus. La personne examine enfin si le ticket traduit correctement la demande.

La proposition ne doit pas être confondue avec la preuve de création. Le système de tickets retourne un identifiant et un statut qui établissent le résultat de l'opération. Un texte de l'agent affirmant « ticket créé » ne remplace pas cette réponse vérifiée.

Prévoir les réponses impossibles à accepter

Un résultat peut être incomplet, invalide, contradictoire ou insuffisamment étayé. Décidez si l'application doit demander une précision, présenter un brouillon, refaire une tentative bornée ou arrêter le traitement. La possibilité de signaler qu'un élément manque fait partie du contrat de sortie.

L'objectif est de rendre l'intégration contrôlable malgré la variabilité du modèle. Des validations déterministes peuvent encadrer certains aspects ; elles ne rendent pas l'ensemble du comportement déterministe. Les essais doivent donc examiner aussi la répétition d'un même cas et les réactions aux entrées difficiles.

À consigner. Le format, les contrôles de structure et de contenu, les informations absentes acceptables et les réactions aux sorties rejetées.

Question de contrôle. Votre application peut-elle distinguer une proposition valide, une action autorisée et une opération effectivement réalisée ?

Préparer la décision de suite

Le POC doit produire des éléments pour décider de poursuivre, de modifier le périmètre ou d'arrêter. La décision attendue mérite d'être formulée avant les essais. Elle détermine les cas à préparer et les observations à conserver.

Je commencerais par décrire les conditions dans lesquelles l'expérimentation aurait une valeur pour votre organisation. Cela peut concerner l'effort de préparation, le taux de correction, la fiabilité d'une action bornée ou la compatibilité avec certaines contraintes. Les seuils doivent être choisis par l'équipe pour ce contexte.

Construire une évaluation contradictoire

Le jeu d'essai inclut des demandes ordinaires, des informations manquantes et des cas qui doivent être refusés. Pour le support, il faut aussi examiner les tentatives de création sans autorisation, les catégories interdites, les pannes d'outil et les demandes répétées.

Conservez les données utiles à la comparaison : versions, configuration, cas d'essai, résultats et corrections. Si plusieurs exécutions d'un même cas sont nécessaires pour examiner la variabilité, leur nombre et les observations attendues doivent être convenus. Aucun essai fini ne démontre l'absence de tous les échecs futurs.

Présenter un bilan qui autorise un choix

Un bilan rapproche l'hypothèse, les résultats observés et leurs limites. Il distingue les problèmes qui invalident l'approche de ceux qui demandent un travail supplémentaire. Un résultat incomplet doit rester identifiable ; il ne devient pas concluant parce que la démonstration finale a été soigneusement choisie.

Une poursuite peut prendre plusieurs formes : un nouvel essai sur une inconnue précise, une évolution du périmètre ou un travail préparant une mise en exploitation. Les exigences de production restent à examiner : surveillance, gestion des incidents, responsabilités d'exploitation, performances, données et maintenance.

À consigner. La décision attendue, les critères, le jeu d'essai, les résultats observés, les inconnues et la personne habilitée à décider de la suite.

Question de contrôle. Si le résultat est mitigé, saurez-vous expliquer ce qui mérite d'être poursuivi et ce qui doit être corrigé ou abandonné ?

PRÉPARER

Examiner les prérequis sans calculer de score

Une addition de bonnes réponses peut masquer une condition essentielle encore absente. Avant d'exécuter le POC, je vous propose de distinguer les prérequis du cadre d'essai et les sujets dont vous souhaitez précisément apprendre quelque chose.

Les conditions nécessaires au périmètre choisi

Le besoin et la décision attendue doivent avoir un responsable. Les données et les accès utilisés doivent être autorisés. Les effets possibles doivent être délimités et les opérations sensibles doivent disposer des contrôles prévus. L'environnement d'essai doit permettre d'interrompre le travail et de traiter ses conséquences.

Si l'une de ces conditions manque, il faut corriger le cadre ou réduire le périmètre avant l'action qui en dépend. Une expérimentation limitée à la préparation de brouillons peut, par exemple, précéder l'ouverture d'un outil d'écriture. Ce changement de périmètre doit être explicite.

Point à examiner	Élément à préparer
Besoin et hypothèse	Travail actuel, résultat attendu, décision à éclairer
Périmètre et outils	Opérations permises, exclusions et traitement des échecs
Données et environnement	Jeux autorisés, flux, droits et traces
Validation	Personnes habilitées, informations à présenter, parcours de refus
Résultat attendu	Contrat de sortie et contrôles d'acceptation
Évaluation	Cas d'essai, mesures et critères convenus

Identifier ce qui reste à apprendre

La qualité des propositions, l'effort de correction ou le comportement d'un modèle peuvent rester incertains : ce sont des objets possibles de l'expérimentation. Une incertitude sur le droit d'utiliser les données ne se traite pas de la même manière.

Pour chaque point, notez « Prêt », « À clarifier » ou « Hors périmètre décidé », avec un fait observable et une action. Indiquez qui doit agir et quelle opération attend cette clarification. Cette grille oriente la préparation ; elle ne certifie ni une aptitude à la production ni une conformité générale du système.

ÉVITER LES ERREURS

Les sept erreurs à éviter

1 Choisir un agent avant de définir le besoin

Une recherche documentaire, un formulaire assisté ou un traitement classique peuvent suffire. Examinez le travail et son point de comparaison avant de retenir une capacité agentique.

2 Exposer un outil trop large ou trop tôt

Un accès générique donne davantage de possibilités que l'opération métier attendue. Délimitez les lectures comme les écritures et activez les capacités au rythme des contrôles disponibles.

3 Prendre une confirmation pour une autorisation complète

Un clic ne règle ni les droits de la personne ni la validité de l'action. L'accord doit être compréhensible, porter sur les paramètres concernés et être appliqué avant l'exécution.

4 Accepter une sortie parce que sa structure est correcte

Un JSON valide peut contenir une catégorie interdite ou un fait inventé. Préparez les contrôles métier et la manière de rejeter un résultat, même bien formé.

5 Mélanger les données et les instructions

Une demande, un document ou une réponse d'outil peuvent contenir un texte hostile. Leur contenu ne doit pas acquérir le statut d'instruction autorisée. Les permissions et contrôles d'exécution limitent les effets possibles.

6 Confondre abstraction et interchangeabilité

Un client commun facilite une évolution. Vérifiez les capacités et les résultats avant de conclure que deux modèles ou deux fournisseurs peuvent être substitués dans votre cas.

7 Terminer par une démonstration sans bilan

Une séquence réussie ne résume pas les essais. Présentez aussi les refus, les échecs, les reprises et les inconnues. La décision de suite doit porter sur les résultats observés dans leur périmètre.

Ces erreurs peuvent se renforcer : un outil trop permissif, associé à une validation imprécise, peut rendre un résultat convaincant difficile à accepter. Leur traitement fait partie du cadrage et de l'évaluation.

UN CAS CONCRET

Une fiche de cadrage sur le cas de support

Ce cas fictif illustre la manière de formaliser un POC. Les critères chiffrés et les conditions détaillées de mesure doivent être définis avec l'équipe avant le démarrage ; les formulations ci-dessous ne constituent pas un protocole d'essai complet.

Le besoin et la décision attendue

Des gestionnaires préparent des tickets à partir de demandes de support. L'équipe veut décider si un agent peut les aider à produire une proposition exploitable, avec un effort de correction acceptable et une création explicitement autorisée. La comparaison porte sur la pratique actuelle et une option plus simple.

Le périmètre autorisé

L'agent consulte uniquement les informations et catégories accessibles à l'utilisateur. Il prépare les champs du ticket. La création se fait dans un environnement d'essai, après confirmation des paramètres par une personne habilitée. La fermeture de tickets, la modification des droits et l'envoi au client restent exclus.

Les données et les composants

Les essais utilisent des demandes fictives et des cas représentatifs préparés pour l'exercice. Le modèle, le fournisseur, les versions et la configuration sont identifiés. Les accès aux outils et les informations conservées dans les traces sont documentés. MAF sert de socle d'intégration pour la variante étudiée.

Les contrôles à effectuer

L'équipe examine les champs produits, les corrections humaines et les temps de traitement. Elle vérifie les parcours d'accord, de refus et d'interruption. Elle teste aussi les outils indisponibles, les demandes répétées et les tentatives d'action hors périmètre. Les règles de décision doivent préciser quels résultats sont acceptables et quels échecs imposent une correction avant la suite.

Le bilan à présenter

Le responsable métier et l'architecte examinent les résultats avec les limites du jeu d'essai. Ils décident de poursuivre, de réduire le périmètre ou d'arrêter. Une poursuite identifie les travaux nécessaires et leur responsable. Elle ne vaut pas, à elle seule, autorisation de déploiement en production.

Je vous conseille de conserver cette fiche avec les résultats, les versions et les décisions. Elle permettra de comprendre ce qui avait été cherché, même plusieurs mois après la fin de l'expérimentation.

APPROFONDIR AVEC MAF

Utiliser MAF pour approfondir la pratique

MAF apporte des mécanismes pour construire des agents et les intégrer à des applications .NET. Leur choix et leur configuration doivent être examinés pour le scénario retenu. Un framework facilite l'implémentation ; les exigences métier, les permissions et les critères de décision restent à définir par votre équipe.

Reprendre les cinq exemples d'initiation

La vidéo « Initiation à la programmation Agentique en C# » et son archive de sources proposent une progression depuis un échange simple jusqu'à une sortie structurée. Chaque exemple permet d'observer un mécanisme, puis d'identifier ce qu'il faudrait compléter pour votre contexte.

Exemple	Question à examiner
Conversation de base	Quel service apporte la réponse au besoin visé ?
Outil passif	Quelle information réelle est obtenue et avec quels droits ?
Outil actif	Quelle opération peut être déclenchée et dans quelles limites ?
Validation humaine	Qu'est-ce qui est présenté, autorisé ou refusé ?
Sortie structurée	Que peut contrôler et consommer l'application ?

Vidéo d'initiation : https://youtu.be/0pdswhV3_Uo

Sources des exemples : <https://e-naxos.com/Download/Demos-Sources-Microsoft-Agent-Framework.zip>

Cours MAF en français et en anglais : <https://www.e-naxos.com/maf/>

Conserver les versions de vos essais

Les exemples constituent un support pédagogique. Vérifiez leurs prérequis et leur correspondance avec les bibliothèques que vous utilisez. Conservez les versions et la configuration de votre propre expérimentation ; une mise à jour peut demander de réexaminer les résultats.

La documentation officielle et les exemples actuels du framework complètent les ressources d'initiation. Les références utiles sont indiquées à la page suivante. Ce guide de cadrage reste volontairement indépendant d'une séquence précise d'installation ou de noms d'API susceptibles d'évoluer.

POUR POURSUIVRE

Choisir la suite adaptée au besoin

Si votre cas d'usage reste flou, un atelier de décision et d'architecture peut aider à préciser le besoin, les contraintes et ce qui mérite d'être essayé. Si les résultats existent déjà, une revue de trajectoire permet d'examiner ce qui justifie une poursuite, une correction ou un arrêt.

Je peux aussi intervenir sur l'intégration d'IA dans une application .NET, l'architecture, le développement ciblé ou la transmission des compétences. Les cours et ateliers MAF s'adaptent au niveau de l'équipe et au travail concerné. Nous convenons du périmètre et du résultat attendu avant l'intervention. La préparation d'une mise en production demande une évaluation de ses exigences propres.

Distinguer le produit et sa fabrication

Si votre question porte sur la manière dont vos équipes utilisent des agents pour développer, le livre blanc « Développer avec les agents IA en gardant la maîtrise du projet » propose une autre entrée : spécifications, architecture, validation et mémoire du projet. Il est accessible depuis les ressources du site e-naxos.com.

SAW, pour SDD Another Way, permet de structurer ce travail de développement. Sa présentation et ses spécifications sont accessibles sur <https://www.e-naxos.com/saw/>. L'intégration d'un agent dans un logiciel et l'organisation d'agents de développement peuvent ainsi être traitées chacune avec leur périmètre.

Références techniques

Documentation MAF : <https://learn.microsoft.com/en-us/agent-framework/>

Fournisseurs de modèles : <https://learn.microsoft.com/en-us/agent-framework/agents/providers/>

Approbation des outils : <https://learn.microsoft.com/en-us/agent-framework/agents/tools/tool-approval>

Sorties structurées : <https://learn.microsoft.com/en-us/agent-framework/agents/structured-outputs>

Les références techniques ont été consultées pour cette édition d'octobre 2026. Pour choisir les API et les options de votre réalisation, utilisez la documentation correspondant aux versions effectivement retenues.

Parler de votre cas d'usage

Olivier Dahan — expert indépendant C# et .NET, architecte, auteur et formateur, Microsoft MVP depuis 2009. J'interviens personnellement ; E-Naxos SARL porte cette activité.

Décrivez l'application ou le processus concerné, l'équipe et la décision à prendre : odahan@e-naxos.com

<https://e-naxos.com/#contact>